A. Calibration Algorithms and Metrics

In this section, we further discuss in detail the various families of approaches commonly used to improve and measure neural network calibration.

A.1. Entropy-based Methods

Entropy-based methods have played an important role in calibrating deep neural networks, as maximizing the entropy helps penalize overconfident predictions [47, 53, 59]. As mentioned in the main text, naively penalizing all predictions can cause underconfident predictions. While various works have proposed different approaches in controlling the entropy term, the Focal Loss [15, 37, 47] and it's variants offer adaptive/automated mechanisms in obtaining suitable values of γ for each sample.

While these automated mechanisms tend to help with ID calibration, many works fail to acknowledge the importance of OOD calibration since the parameters obtained during training/validation may not work during testing [57]. As a work-around, we find that entropy-based methods can be extended to include OOD Maximum Entropy constraints [26, 53] or Dual logit manipulation [67], showcasing the versatility of entropy-based methods. Since these methods all share the form of the Focal loss, we can easily pair all of them together into a single step.

A.2. Regularizers

Mixup is an effective regularization technique that augments [81] both input features and labels. Mixup works particularly well on both wider and deeper networks [83] and can be particularly useful in improving network calibration [7, 68]. As an extension to vanilla Mixup, RankMixup [56] can be used to ensure that the augmented samples have lower confidences than the original samples.

A.3. Margin-based Methods

Margin-based methods tend to restrict model confidences by a constant margin/factor. For example, label smoothing (LS) [48] softens the targets using a constant factor ϵ. Mathematically, the smoothed label s_i is acquired after uniformly adjusting the target $s_i = (1-\epsilon)y_k + \frac{\epsilon}{K}$, which is then used to train the network. Although vanilla LS can be used to improve miscalibration, imposing a constant smoothing factor for all training labels can lead to under-confident predictions. Furthermore, searching for a suitable ϵ is computationally expensive as it requires a grid-search across multiple models during the training phase.

Instead of implementing a fixed constant, several works have been proposed to adaptively or conditionally approximate the label smoothing function during training. For example, MDCA [24] utilizes a regularization term, which enforces predicted confidences to be as close to the average accuracy as possible. This can lead to a parabolic smoothing function [58], that is adaptively dependent on the predicted confidences. Which can be problematic, since both high and low confidence predictions are weakly penalized. Another approach would be to only conditionally smooth predictions based on a margin. For instance, MBLS [40] and CALS-ALM [41] propose to restrict output logits by a user defined margin, but can be sensitive to hyper parameter settings. CRL [46] ordinarily ranks predictions based on the number of times each sample is predicted correctly, however it requires a buffer to store the correctness history. Which can be empty during the earlier stages of training and idle during later phases when the model's accuracy is high.

Building upon these methods, Adaptive Conditional Label Smoothing (ACLS) [58] aims to dynamically approximate the label smoothing function.

$$\mathcal{L}_{\text{ACLS}} = \begin{cases} \lambda_1 \max(0, h_k^\theta(x) - \min_k(h_k^\theta(x)) - m_{\text{A}})^2 & \text{(a)} \\ \lambda_2 \max(0, h_{\hat{y}}^\theta(x) - h_k^\theta(x) - m_{\text{A}})^2 & \text{(b)} \end{cases} \tag{15}$$

For case (a) $k = \hat{y}$, the smoothing function is directly proportional to $h_k^\theta(x)$, thereby lowering confidences. Similarly, when $k \neq \hat{y}$ in case (b), the effects of the smoothing function decreases, allowing the logits and confidences to increase. m_{A} denotes the ACLS margin and λ_1, λ_2 are hyperparameters for cases when $k = \hat{y}$ and $k \neq \hat{y}$.

By adopting a smoothing and indicator function, the Adaptive Conditional Label Smoothing (ACLS) [58] method seeks to combing the benefits of both adaptive and conditional methods without the use of an additional correctness history.

A.4. Post-hoc Processing

The fundamental idea behind post-hoc processing methods is to obtain a mapping function/temperature that modifies the model's logits thus changing it's predicted confidence. The most popular post-processing step is the vanilla temperature scaling (TS) [60], which manipulates the model's confidences without changing the final class label predictions. For example, a value of $T < 1$ leads to a lower entropy or "peaky" distributions and a value of $T > 1$ gives higher entropy or "flatter" predictions.

The typical approach in obtaining the temperature parameter, is to minimize the *average* calibration error or NLL over a seperate valdation set. While vanilla TS has been found to be effective in reducing network over-confidence [19], it generally reduces the confidence of every sample - even when predictions are correct. Other forms of post-hoc processing include calibration using, model ensembles [82], splines [21] and distribution matching [34, 71]. For our post-processing step, we use AdaTS since it is the SOTA method for post-processing methods and adaptively chooses a samplewise temperature for scaling model predictions.

A.5. Calibration Metrics

Expected Calibration Error (ECE): The ECE is the most widely used metric in the literature and directly tied to the definition of calibration [19, 69]. By splitting the predicted confidences in B evenly separated bins, each containing n_b samples. The ECE is then simply a scalar measuring the weighted errors between the *acc* and *conf* of each bin [52]: ECE $= \sum_{b=1}^{B} \frac{n_b}{N} |acc(b) - conf(b)|$. Despite the ECE's popularity, many recent works have pointed out the limitations of the ECE, such as bin size sensitivity and it's lack of consideration for classwise calibration. For a fair and thorough analysis, we introduce other calibration metrics that cover the weaknesses of the ECE.

Classwise ECE (CECE): As most calibration metrics typically only considers the max confidence probabilities, the CECE considers the macro-averaged ECE of all K classes. Predictions are binned individually for each respective class and the calibration error is measured for each class level bin [55]. CECE $= \frac{1}{K} \sum_{b=1}^{B} \sum_{k=1}^{K} \frac{n_{b,k}}{N} |\text{acc}(b,k) - \text{conf}(b,k)|$.

Overconfidence Error (OE): For safety-critical applications, overconfident mispredictions are potentially hazardous. The OE penalizes overconfident bins that have higher confidences than accuracy [68]: OE $= \sum_{b=1}^{B} \frac{n_b}{N} \Big[conf(b) \times \max(conf(b) - acc(b), 0) \Big]$.

Kolmogorov-Smirnov Error (KSE): As many calibration metrics are often sensitive to the number of B bins used during the partitioning of empirical distributions. The KSE[21] is a bin-free alternative that numerically approximates the differences between two empirical cumulative distributions. The KSE for top-1 classification is given as the following integral, with z_k denoting the predicted probabilities: KSE $= \int_0^1 |P(k|z_k) - z_k| P(z_k) dz_k$.

Adaptive ECE (AdaECE: as the ECE is known to be biased towards higher confidence bins, the AdaECE [54] is proposed to adaptively/evenly measure samples across bins: AdaECE $= \sum_{b=1}^{B} \frac{n_b}{N} |\text{acc}(b) - \text{conf}(b)|$ s.t. $\forall b, i \cdot |B_b| = |B_i|$.

Negative Log-likelihood (NLL): Commonly referred to as cross entropy in deep learning. The NLL [22] measures the alignment between a model's confidence $P_i(y_k|x)$ and targets y_k: NLL $= -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_k \log P_i(y_k|x)$.

Hyperparameters	Values
Learning rate η	0.1
Batch size	512 or 256
Optimizer	SGD or Adam
Scheduler	Cosine Annealing or Fixed
Epochs	200 or 50
Margin m_{ACLS}	6.0
Mixup α	1.0
Mixup margin m_{MRL}	2.0
γ starting	1.0
γ max	20.0
γ min	-2.0
No. of bins B	15.0
Learning rate for attention block	3e-4

Table 4. Hyperparameters used for optimizing *Peacock*

B. Implementation Details for Peacock

B.1. Algorithm Details and Hyperparameters

For our implementation of *Peacock*, we first select the mean constraint for of MaxEnt loss as our starting algorithm and compute Lagrange multipliers λ_n using the Newton Raphson method. This step is performed in $\mathcal{O}(n)$ time using the helper function $g(\lambda)$ and its derivative $g'(\lambda)$ before model training begins.

For each iteration, the pairwise 1v1 constraints of CPC loss are first computed before incorporating the adaptive γ selection mechanism of AdaFocal loss. This step also includes the second highest confidence $P_i(y_j|x)$ from Dual Focal loss to AdaFocal loss. This way we can reduce compute overhead by combining both calibration methods into a single step: $\mathcal{L}_{\text{AdaFocal}}^{\text{Dual}} = \mathcal{L}_{\text{AdaFocal}} + \mathcal{L}_{\text{Dual}}$.

Next, RankMixup is performed for every sampled input image and label, with the MRL loss computed using the coefficients α and m. For texts datasets, we perform RankMixup at the feature level. Although vanilla Mixup has been found to hurt ID calibration performance [74], our findings suggest that by combining RankMixup with other algorithms good balance between ID and OOD calibration can be achieved. In our experience, we find that a large ACLS margin, can lead to numerical instability when the number of classes is large, thus we fixed $m_{\text{ACLS}} = 6.0$.

For completeness, we include the ACLS step in Algorithm 1, however in our ablation study we show that ACLS does not improve overall calibration performance and is not included during optimization or our final proposed version of *Peacock*. Next, an optional post-processing step using AdaTS is performed by learning the adaptive temperature on a seperate validation set. Finally, for the importance weighted form of *Peacock*, we randomly initialize a self-attention block and optimize it with Eq. (14) with Adam optimizer and a learning rate of 3e-4 to learn a set of importance weights for each loss term.

Algorithm 1: Peacock - Unified Multi-Objective Optimization Calibration Framework

Data: Given training and validation set $D_{\text{train}} = (x_i, y_i)_{i=1}^{N}$, $D_{\text{val}} = (x_v, y_v)_{v=1}^{V}$

Initialize neural network parameters $\boldsymbol{\theta}$, learning rate schedule η and uniformly distributed weights $w_t = \frac{1}{A}$
Compute the global and local expectations for the mean and variance constraints μ, σ^2
    $\hookrightarrow$ $\mathbb{E}[\mathcal{Y}] = \mu$ and $\mathbb{E}[\mathcal{Y}^2] = \sigma^2$
Solve numerically for $\lambda_\mu \leftarrow$ `NewtonRaphson()` // MaxEnt loss root-finder
for $e \in epochs$ **do**
  for $i \in B$ **do** // Sample mini-batch of size B
    Perform FastMixup on images: $\tilde{x} = \beta x_i + (1-\beta)x_j$ // RankMixup
    Perform FastMixup on labels: $\tilde{y} = \beta y_i + (1-\beta)y_j$ // RankMixup
    Compute 1v1 loss: $\mathcal{L}_{\text{CPC}}^{\text{1v1}} = -\frac{1}{(C-1)} \sum_{j \neq y} \log \frac{P_i y}{P_i y + P_i j}$ // CPC loss
    if $\gamma_{t,b} \geq 0$ **then**
      $\mathcal{L}_{\text{AdaFocal}}^{\text{Dual}} = -\sum_k (1 - P_i + P_j)^{\gamma_{t,b}} \log P_i$ // Dual AdaFocal loss
    else if $\gamma_{t,b} < 0$ **then**
      $\mathcal{L}_{\text{AdaFocal}}^{\text{Dual}} = -\sum_k (1 + P_i + P_j)^{|\gamma_{t,b}|} \log P_i$ // Inverse Dual AdaFocal loss
    Compute MaxE loss $\mathcal{L}_{\text{ME}} = \lambda_\mu (\sum_k f(\mathcal{Y}) P_i(y_k|x) - \mu_G + \sum_k f(\mathcal{Y}) P_i(y_k|x) - \mu_{Lk})$ // MaxEnt loss
    Compute MRL loss $\mathcal{L}_{\text{MRL}} = \max(0, \max_k \tilde{P} - \max_k P + m_{\text{MRL}})$ // RankMixup
    if $j = \hat{y}$ **then**
      $\mathcal{L}_{\text{ACLS}} = \lambda_1 \max(0, g_j^\theta(x) - \min_k(g_k^\theta(x)) - m_{\text{ACLS}})^2$ // ACLS regularizer
    else if $j \neq \hat{y}$ **then**
      $\mathcal{L}_{\text{ACLS}} = \lambda_2 \max(0, g_{\hat{y}}^\theta(x) - g_j^\theta(x) - m_{\text{ACLS}})^2$ // ACLS regularizer
    $w_t =$ `ImportancePeacock(`$\mathcal{L}_t(\boldsymbol{\theta})$`)` // Compute importance loss weights

    Compute Peacock:
    $\hookrightarrow$ $\mathcal{L}_{\text{Peacock}} = \mathcal{L}_{\text{AdaFocal}}^{\text{Dual}} + w_1 \mathcal{L}_{\text{constraints}}^{\text{ME}} + w_2 \mathcal{L}_{\text{CPC}}^{\text{1v1}} + w_3 \mathcal{L}_{\text{MRL}}$
    $\boldsymbol{\theta}_{\text{new}} \leftarrow \boldsymbol{\theta}_{\text{old}} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{Peacock}}$ // Update parameters $\boldsymbol{\theta}$ by gradient descent
return $\boldsymbol{\theta}$

Apply temperature scaling: $\boldsymbol{\theta}_{\text{AdaTS}} \leftarrow$ `AdaptiveTS(`$D_{val}, \boldsymbol{\theta}$`)` // AdaTS
Function `NewtonRaphson()`**:**
    $\delta =$ 1e-15 // A small tolerance or stopping condition
    while $g(\lambda) > \delta$ **do**
      $\lambda_{n+1} = \lambda_n - \frac{g(\lambda)}{g'(\lambda)}$ // Update Lagrange Multipliers λ_n
    return λ_n
Function `ImportancePeacock()`**:**
    $\min_{w_t} \left\{ \left|\left| \sum_{t=1}^{A} w_t \sqrt{\frac{\Delta_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta})}{\eta}} \right|\right|_2^2 \middle| \sum_{t=1}^{A} w_t = 1, w_t \geq 0 \quad \forall t \right\}$
    return w_t
Function `AdaptiveTS(`$D_{val}, \boldsymbol{\theta}$`)`**:**
    Initialize VAE and MLP parameters $\boldsymbol{Q}, \boldsymbol{\phi}$
    while $t <$ *steps* **do**
      for $v \in B$ **do** // Sample mini-batch of size B
        $\nabla_{VAE} \leftarrow \nabla \mathbb{ELBO}[\Phi(x)]$
        $\tilde{q} = \{\log P(z|y) | \forall y\} \quad z \sim Q_\phi(z|x)$
        $\nabla_T \leftarrow \log(\text{softmax}(g_\theta / T))$
        $(\boldsymbol{\theta}, \phi)_{t+1} \leftarrow (\boldsymbol{\theta}, \phi)_t - \alpha_{\text{lr}}(\nabla_{VAE} + \nabla_T)$
    return $\boldsymbol{\theta}_{\text{AdaTS}}$

Hyperparameters In general, we try to keep the default settings of each algorithm. However, when trying to combine multiple of these components, it may become inevitable for some tuning to be performed. Indeed, performing a grid-search would be the best way to obtain the optimal hyper-parameters. However, as discussed in our *Limitations*, the number of parameters scale exponentially with the number of calibration components selected for optimization. This can be easily become very compute intensive and would not be the focus of our work.

B.2. Accelerating RankMixup

Fig. 7 illustrates the comparisons between the original RankMixup method and the optimized version proposed in our paper. RankMixup, in its original form, requires two forward passes during training: one for a full minibatch (e.g., 512) of original images and another full minibatch of mixed images. This process can be computationally expensive, especially for large datasets or complex models. As a workaround, we propose an optimized variant of FastRankMixup, which addresses this limitation by dividing a full batch of images into two halves: containing a minibatch of half original and half mixed images (e.g., 512 ÷ 2 = 256).

This way, we only require a single forward pass instead of the two forward passes, delivering a 2x speedup during training compared to the original RankMixup implementation. This improvement in training efficiency can be particularly beneficial for large-scale training tasks, where computational resources are often constrained. A caveat to this method is that the minimum batchsize required will always be two, as at least two samples are needed to be paired together for Mixup to be performed.

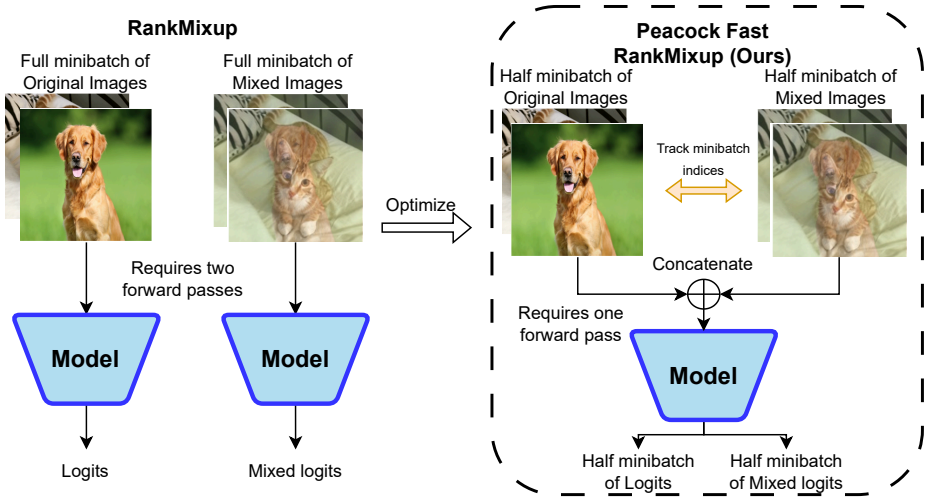


Figure 7. During training, RankMixup requires two forward passes: one for original images and one for mixed images, in order to compute $\mathcal{L}_{MRL}$. We optimize RankMixup by mixing images and labels batchwise, resulting in a 2x speed up during training.

C. Proofs

C.1. Temperature-scaled bounds

Consider a temperature/mapping function T which scales the output logits/hypothesis $h^{\boldsymbol{\theta}}$ of a model. Then the average of each temperature scaled hypothesis is given as:

$$\overline{\text{CE}}^2(\mathcal{T}(h^{\boldsymbol{\theta}})) = \frac{1}{A}\sum_{t=1}^{A}\text{CE}^2(\mathcal{T}(h_t^{\boldsymbol{\theta}})) \tag{16}$$

Considering equal contributions of each individual temperature scaled hypothesis, the temperature scaled multi-objective learner $\mathcal{T}(H^{\boldsymbol{\theta}})$ has the expected squared CE:

$$\text{CE}^2(\mathcal{T}(H^{\boldsymbol{\theta}})) = \mathbb{E}\left[\left(\frac{1}{A}\sum_{t=1}^{A}\text{CE}(\mathcal{T}(h_t^{\boldsymbol{\theta}}))\right)^2\right] \tag{17}$$

which follows the same bounds as previously defined in the main paper.

$$\text{CE}^2(\mathcal{T}(H^{\boldsymbol{\theta}})) \leq \overline{\text{CE}}^2(\mathcal{T}(h^{\boldsymbol{\theta}})) \tag{18}$$

Empirically, Tab. 1 demonstrates that if the same mapping function or temperature $\mathcal{T}$ is applied to each hypothesis (e.g., AdaTS), then the average of the scaled combined learner will also obey the upper bound of the above inequality.

C.2. Estimating the Gradient

Recall in Sec. 4.2 of our main paper, the direct computation of $\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})$ requires the use of retaining the computational graph[4] after the backward pass, which can be compute intensive and significantly slows down training time.

[4] For more details, see Pytorch autograd framework: *https://pytorch.org/docs/stable/autograd.html*

In this section, we demonstrate that decrease rate estimates for each loss term can act as alternatives to direct gradient recomputation. By simply storing the previous loss value computed (single step look-back), we can avoid graph retention during the optimization for w_t. Using a simple example, we also show that our decrease rate estimates are closely related to the solutions obtained using gradient descent. For simplicity, we denote the partial derivatives as $\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta}) = \frac{\partial\mathcal{L}_t(\boldsymbol{\theta})}{\partial(\boldsymbol{\theta})}$ and the difference between old and new parameters as $\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})$.

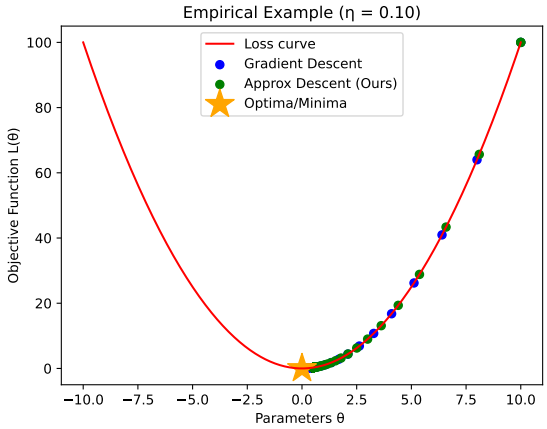
Consider the following task of approximating $\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})$. By using the first order form of Taylor's Theorem, the loss gradients can be rewritten as the following equation:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta}) = \frac{\mathcal{L}_t(\boldsymbol{\theta}_{\text{new}}) - \mathcal{L}_t(\boldsymbol{\theta}_{\text{old}})}{\Delta\boldsymbol{\theta}} + \epsilon(\boldsymbol{\theta}) = \frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{\Delta\boldsymbol{\theta}} + \epsilon(\boldsymbol{\theta}) \tag{19}$$

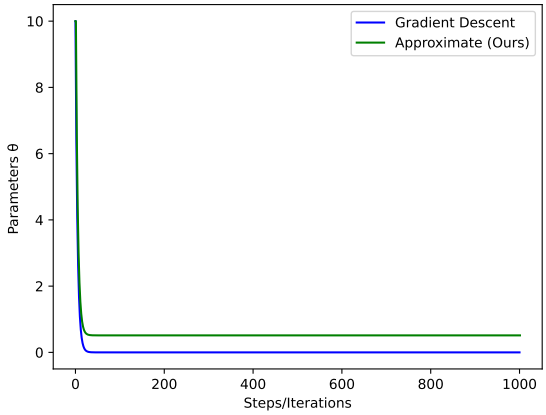
where $\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})$ is the rate of change for each loss term with respect to the change of model parameters $\boldsymbol{\theta}_{\text{new}}$ and $\boldsymbol{\theta}_{\text{old}}$, paired by a small error term $\epsilon(\boldsymbol{\theta})$. From the gradient descent update rule, the change in model parameters is given by:

$$\begin{aligned}\boldsymbol{\theta}_{\text{new}} &= \boldsymbol{\theta}_{\text{old}} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta}) \\ \Delta\boldsymbol{\theta} &= -\eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})\end{aligned} \tag{20}$$

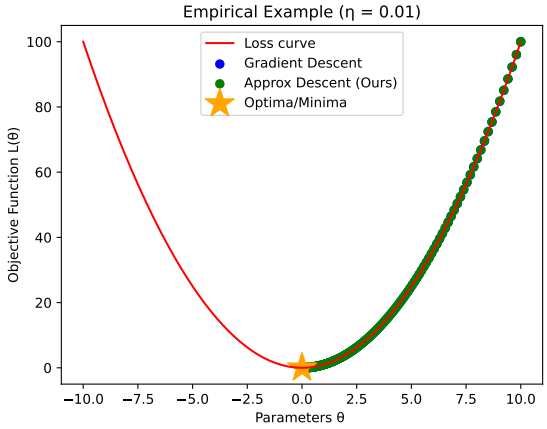
where the difference between new and old network parameters are obtained using the gradients and a learning rate η. By substituting Eq. (20) into Eq. (19):



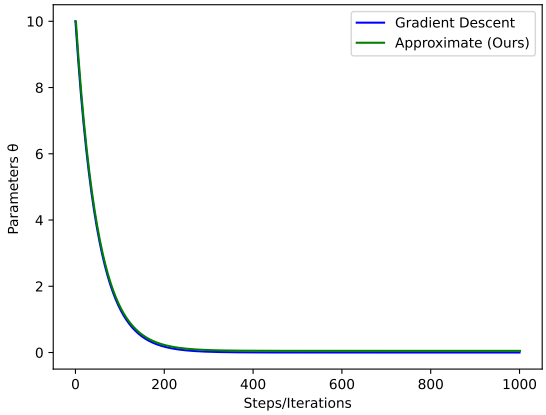
(8a) Toy sketch illustrating the solution of our method (green) compared to gradient descent (blue). Our method closely approximates gradient descent.



(8b) We compare the solutions given by our method and gradient descent, for $\eta = 0.1$, our solution is close to the solution by gradient descent.



(9a) By reducing the learning rate, our method (green) provides an even closer estimate to gradient descent (blue).



(9b) We compare the solutions given by our method and gradient descent for $\eta = 0.01$, our method can be improved by reducing the learning rate.

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})^2 = \frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{-\eta} + \epsilon(\boldsymbol{\theta}) = \frac{\mathcal{L}_t(\boldsymbol{\theta}_{\text{old}}) - \mathcal{L}_t(\boldsymbol{\theta}_{\text{new}})}{\eta} + \epsilon(\boldsymbol{\theta})$$

**Note the flip in sign*

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta}) = \sqrt{\frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{\eta} + \epsilon(\boldsymbol{\theta})} \approx \sqrt{\frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{\eta}} \tag{21}$$

with the small error term $\epsilon(\boldsymbol{\theta})$ dropped.

Key Assumptions: Our main paper highlighted the essential assumptions underlying this formulation: 1.) The loss terms $\mathcal{L}_t$ are convex and optimizable by gradient descent. 2.) Each loss term monotonically decreases i.e., the loss evaluated at previous iterations will always be strictly larger than the loss at the current iteration $\mathcal{L}_t(\boldsymbol{\theta}_{\text{old}}) > \mathcal{L}_t(\boldsymbol{\theta}_{\text{new}})$. This assumption ensures that the ratio $\sqrt{\frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{\eta}}$ remains positive, avoiding the computation of complex numbers. Moreover, the small learning rates commonly used in deep learning frameworks tend to be sufficiently small (e.g., η = 2.5e-4) allowing for accurate linear approximations. In practice, we can apply the ReLU function to the gradient update, i.e $\sqrt{\text{ReLU}(\frac{\Delta_{\boldsymbol{\theta}}\mathcal{L}_t(\boldsymbol{\theta})}{\eta})}$ if the gradient descent step leads to an increase in the loss, violating the assumption that $\mathcal{L}_t(\boldsymbol{\theta}_{\text{old}}) > \mathcal{L}_t(\boldsymbol{\theta}_{\text{new}})$. This ensures that the update is scaled down or ignored in the optimization process.

Simple Empirical Example We further support our findings by including a simple empirical example comparing gradient descent and our proposed method. Consider a smooth, convex objective function $\mathcal{L}_\theta = \theta^2$. Fig. 8a il-

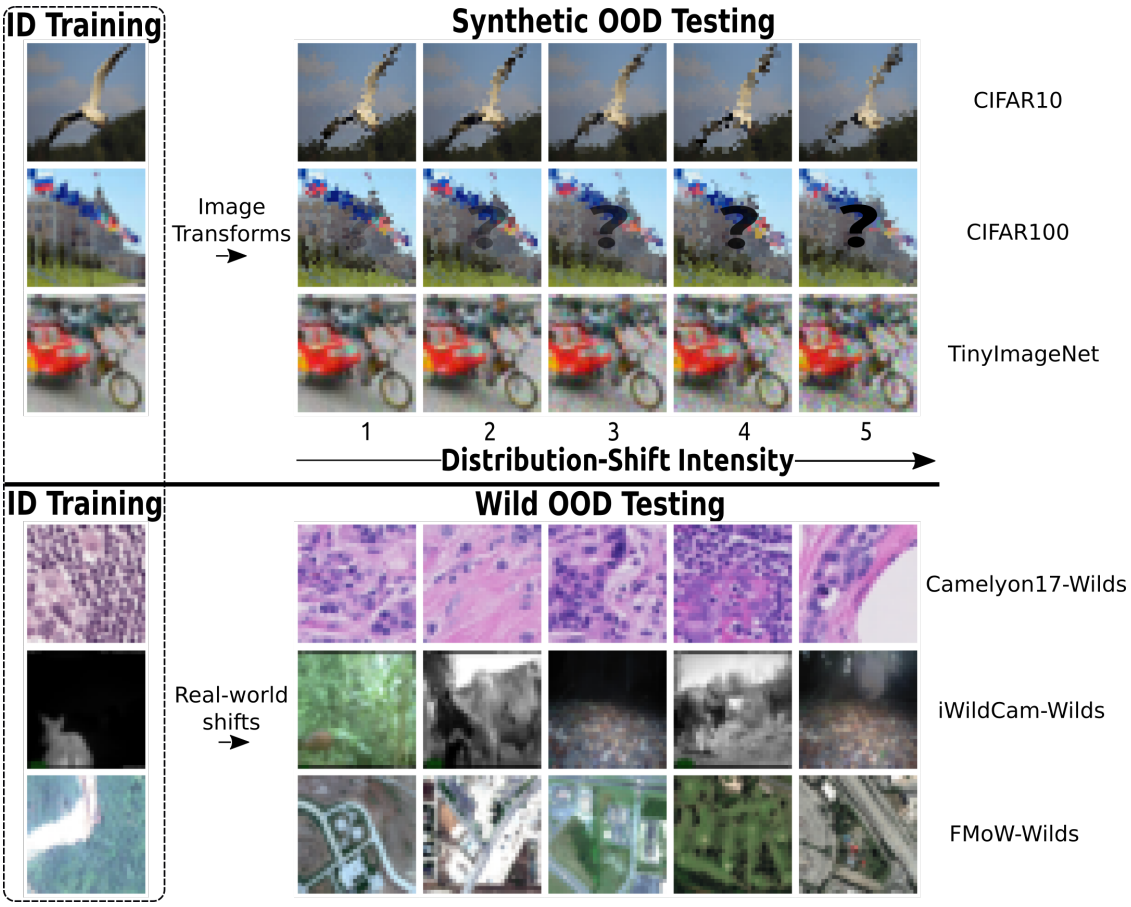


Figure 10. Covariate shifts can be simulated using common image corruptions or caused by natural differences during data collection in-the-wild.

lustrates our goal of obtaining a set of parameters θ such that $\mathcal{L}_\theta$ is minimized. For a fixed learning rate of $\eta = 0.1$, 1000 iteration steps and a starting point of $\theta = 10$, our solution given by our method (in green) is relatively close compared to the solution given by gradient descent (in blue), with a slight delay and an error of roughly 0.5. In Fig. 9a we can further improve our method's solution by reducing the learning rate to $\eta = 0.01$, which provides an even closer estimate to the solutions given by gradient descent and a reduced relative error of roughly 0.05.

D. Supplementary Experiments and Results

D.1. Dataset Details

Synthetic OOD We train our models with clean images from the original CIFAR, TinyImageNet and evaluate their OOD performance on their corrupted forms CIFAR-C, TinyImageNet-C.

1. CIFAR10/CIFAR100 [33] RGB images of size (32x32) containing ten and hundred classes. The training/validation/testing sets contain 45,000/5,000/10,000 samples respectively.
2. TinyImagenet [10] A miniature version of the ImageNet dataset containing images of size (64x64) of 200 classes. There are 100,000 images for training and 10,000 images for validation/testing.
3. CIFAR10-C/CIFAR100-C/TinyImagenet-C [25] A widely popular calibration benchmark, containing corrupted variants of CIFAR and TinyImageNet. Standard image corruptions (total of 19) are applied on the original test sets across five increasing levels of severities.

Real-world OOD For wild OOD, we learn our models using the provided ID training sets and OOD sets for validation and testing [32].

1. Camelyon17 [1]: A binary task to detect if a (32x32) cell tissue slide is benign or malignant. The images are collected across different hospitals with equipment that may vary OOD from the training set.
2. iWildCam [2]: Animal species tend to vary across different backgrounds and terrains. The goal is to classify 182

Dataset	Metric	MaxEnt	AdaFocal	RankMixup	CPC	Dual	ACLS	Peacock (Eq.)	Peacock (Impt.)
CIFAR10	Acc. ↑	94.0$_{\pm 0.2}$	93.4$_{\pm 0.4}$	94.1$_{\pm 0.1}$	94.6$_{\pm 0.1}$	94.5$_{\pm 0.1}$	94.3$_{\pm 0.1}$	93.8$_{\pm 0.1}$	93.9$_{\pm 0.2}$
	ECE ↓	1.1$_{\pm 0.1}$	0.8$_{\pm 0.1}$	3.1$_{\pm 0.1}$	2.9$_{\pm 0.1}$	1.3$_{\pm 0.1}$	3.0$_{\pm 0.1}$	0.6$_{\pm 0.1}$	**0.6**$_{\pm 0.1}$
	CECE ↓	0.4$_{\pm 0.1}$	0.3$_{\pm 0.1}$	2.8$_{\pm 0.1}$	2.7$_{\pm 0.1}$	0.4$_{\pm 0.1}$	2.7$_{\pm 0.1}$	0.2$_{\pm 0.1}$	**0.2**$_{\pm 0.1}$
	NLL ↓	249.8$_{\pm 0.4}$	232.7$_{\pm 0.1}$	346.9$_{\pm 0.2}$	394.4$_{\pm 0.2}$	253.7$_{\pm 0.1}$	345.3$_{\pm 0.3}$	**224.4**$_{\pm 0.4}$	224.5$_{\pm 4.1}$
CIFAR100	Acc. ↑	73.8 $_{\pm 0.1}$	75.8 $_{\pm 0.1}$	74.9$_{\pm 0.3}$	74.7$_{\pm 0.1}$	75.4$_{\pm 0.1}$	75.3$_{\pm 0.1}$	74.5$_{\pm 0.4}$	73.5$_{\pm 0.5}$
	ECE ↓	5.4$_{\pm 0.5}$	6.8$_{\pm 0.1}$	4.9$_{\pm 0.1}$	8.8$_{\pm 0.3}$	9.1$_{\pm 0.1}$	4.5$_{\pm 0.3}$	4.1$_{\pm 0.3}$	**3.9**$_{\pm 0.2}$
	CECE ↓	0.2$_{\pm 0.1}$	0.1$_{\pm 0.1}$	2.9$_{\pm 0.1}$	2.3$_{\pm 0.2}$	0.1$_{\pm 0.1}$	1.7$_{\pm 0.1}$	0.1$_{\pm 0.1}$	**0.1**$_{\pm 0.1}$
	NLL ↓	312.7$_{\pm 0.6}$	306.5$_{\pm 2.3}$	348.6$_{\pm 0.7}$	432.2$_{\pm 0.8}$	319.8$_{\pm 0.4}$	346.6$_{\pm 1.0}$	298.3$_{\pm 1.2}$	**283.9**$_{\pm 0.1}$
TinyImageNet	Acc. ↑	63.1$_{\pm 0.3}$	60.8$_{\pm 0.1}$	61.6$_{\pm 0.3}$	65.0$_{\pm 0.3}$	63.2$_{\pm 0.3}$	64.9$_{\pm 0.1}$	61.2$_{\pm 0.1}$	62.3$_{\pm 0.4}$
	ECE ↓	18.2$_{\pm 0.3}$	6.1$_{\pm 0.5}$	5.5$_{\pm 0.3}$	10.3$_{\pm 0.4}$	6.8$_{\pm 0.1}$	5.0$_{\pm 0.3}$	6.2$_{\pm 0.3}$	**3.9**$_{\pm 0.3}$
	CECE ↓	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	0.1$_{\pm 0.1}$	**0.1**$_{\pm 0.1}$
	NLL ↓	322.0$_{\pm 0.5}$	320.5$_{\pm 0.3}$	343.2$_{\pm 1.0}$	358.5$_{\pm 1.6}$	339.2$_{\pm 1.0}$	342.3$_{\pm 0.4}$	333.9$_{\pm 2}$	**324.2**$_{\pm 2.4}$

Table 5. We report the ID test scores (%) for reruns computed across 3 seeds for *Peacock* and its components.

Dataset	Metric	MaxEnt	AdaFocal	RankMixup	CPC	Dual	ACLS	Peacock (Eq.)	Peacock (Impt.)
CIFAR10-C	AdaECE ↓	6.9$_{\pm 0.1}$	6.2$_{\pm 0.4}$	11.5$_{\pm 0.2}$	10.7$_{\pm 0.4}$	7.2$_{\pm 0.2}$	11.5$_{\pm 0.1}$	6.3$_{\pm 0.1}$	**6.2**$_{\pm 0.3}$
	OE ↓	3.9$_{\pm 0.3}$	**3.0**$_{\pm 0.3}$	9.6$_{\pm 0.2}$	9.1$_{\pm 0.4}$	3.8$_{\pm 0.1}$	9.5$_{\pm 0.1}$	3.3$_{\pm 0.1}$	3.5$_{\pm 0.2}$
CIFAR100-C	AdaECE ↓	11.0$_{\pm 0.1}$	13.7$_{\pm 0.3}$	8.4$_{\pm 0.2}$	13.7$_{\pm 0.2}$	15.5$_{\pm 0.1}$	10.2$_{\pm 0.3}$	9.7$_{\pm 0.3}$	**9.6**$_{\pm 0.3}$
	OE ↓	**0.5**$_{\pm 0.1}$	0.7$_{\pm 0.1}$	2.5$_{\pm 0.2}$	1.8$_{\pm 0.1}$	0.7$_{\pm 0.1}$	2.01$_{\pm 0.1}$	1.3$_{\pm 0.1}$	1.6$_{\pm 0.1}$
TinyImageNet-C	AdaECE ↓	12.6$_{\pm 0.3}$	13.9$_{\pm 0.3}$	20.2$_{\pm 0.2}$	16.3$_{\pm 0.2}$	18.8$_{\pm 0.2}$	20.6$_{\pm 0.4}$	**10.4**$_{\pm 0.2}$	10.7$_{\pm 0.2}$
	OE ↓	4.4$_{\pm 0.3}$	4.5$_{\pm 0.3}$	10.4$_{\pm 0.2}$	8.5$_{\pm 0.2}$	9.4$_{\pm 0.2}$	11.1$_{\pm 0.4}$	**2.3**$_{\pm 0.2}$	2.3$_{\pm 0.2}$
Camelyon17	AdaECE ↓	12.3$_{\pm 0.4}$	20.4$_{\pm 0.1}$	22.4$_{\pm 4.6}$	20.1$_{\pm 1.6}$	15.4$_{\pm 2.5}$	19.6$_{\pm 0.2}$	11.7$_{\pm 0.7}$	**9.8**$_{\pm 1.8}$
	OE ↓	10.9$_{\pm 0.8}$	19.6$_{\pm 0.1}$	22.0$_{\pm 4.6}$	19.8$_{\pm 1.6}$	14.3$_{\pm 2.4}$	18.9$_{\pm 0.1}$	10.6$_{\pm 0.4}$	**9.0**$_{\pm 1.6}$
iWildCam	AdaECE ↓	21.0$_{\pm 3.2}$	23.0$_{\pm 0.5}$	25.5$_{\pm 0.7}$	20.3$_{\pm 1.1}$	13.0$_{\pm 2.5}$	20.6$_{\pm 1.8}$	**9.7**$_{\pm 0.3}$	12.6$_{\pm 1.4}$
	OE ↓	14.3±3.6	16.8$_{\pm 0.2}$	20.4$_{\pm 0.8}$	15.5$_{\pm 0.2}$	8.21$_{\pm 1.4}$	15.4$_{\pm 1.2}$	**5.2**$_{\pm 0.2}$	7.9$_{\pm 1.0}$
FmoW	AdaECE ↓	20.0$_{\pm 9.9}$	20.9$_{\pm 8.6}$	41.7$_{\pm 0.1}$	22.4$_{\pm 0.9}$	9.73$_{\pm 0.1}$	21.7$_{\pm 0.2}$	**10.5**$_{\pm 0.2}$	10.6$_{\pm 0.1}$
	OE ↓	13.7±7.7	14.2±7.0	33.7$_{\pm 0.2}$	16.7$_{\pm 0.8}$	**4.78**$_{\pm 0.1}$	14.6$_{\pm 0.2}$	5.5$_{\pm 0.3}$	5.4$_{\pm 0.3}$

Table 6. We report additional OOD test scores (%) for reruns evaluated on both synthetic and wild benchmarks for *Peacock* and its components.

Algorithm	Acc (%)	ECE (%)	Speed (Sec)	w1	w2	w3	$\sum_t^A w_t = 1$
Equal-Impt.	51.8$_{\pm 0.1}$	9.6$_{\pm 0.2}$	50.1$_{\pm 0.2}$	0.33	0.33	0.33	Yes
MTAN [43]	50.8$_{\pm 0.1}$	10.2$_{\pm 0.4}$	50.7$_{\pm 0.2}$	0.33	0.33	0.33	Yes
CoVV [17]	52.6$_{\pm 0.1}$	12.0$_{\pm 0.4}$	50.6$_{\pm 0.2}$	0.01	0.52	0.47	Yes
GradNorm [5]	48.9$_{\pm 0.6}$	9.3$_{\pm 0.7}$	85.8$_{\pm 0.7}$	2.99	0.00	0.00	No
MT-MOO [65]	51.9$_{\pm 0.5}$	9.3$_{\pm 0.5}$	67.5$_{\pm 0.5}$	0.36	0.47	0.16	Yes
Weighted-Impt. (Ours)	52.6$_{\pm 0.1}$	9.3$_{\pm 0.3}$	50.5$_{\pm 0.2}$	0.00	0.51	0.49	Yes

Table 7. Comparisons of different multi-objective optimization methods for *Peacock*. Our weighted importance formulation is fast and effective.

animal classes collected from camera traps deployed in different areas of the wilderness.

3. FMoW [8]: Satellite imagery of topographies and buildings alike tend to differ greatly across countries. The task is classify the OOD shifted terrains from one out of 62 classes.

D.2. Supplementary Experiments and Results

ID Results: As demonstrated in Tab. 5, our synthetic benchmark results confirm *Peacock*'s highly competitive performance on ID test sets. As discussed in the main text, *Peacock*'s calibration error is inherently bounded by the average calibration error of its constituent components. Consequently, even if some components underperform, *Peacock*'s overall calibration remains well-calibratied, irrespective of whether the data is in-distribution (ID) or out-of-distribution (OOD).

Additional OOD Results: Tab. 5, shows OOD supplementary results evaluated using AdaECE [54] and OE [68]. Our analysis using these additional metrics aligns with the results presented in our primary findings. While the theoretical proofs in our main paper and Appendix C are explicitly stated only for the ECE, we anticipate that our arguments remain valid for other calibration metrics, which are often derivatives or closely related to ECE. We intend to explore this aspect further in future research. Additional results for non-standard OOD scenarios are presented in Tab. 3.

Additional Multi-Objective Optimization Results: Additional results for our proposed weighted-importance formulation are provided in Tab. 8 and Tab. 9. Our results highlight the versatility, effectiveness and speed acorss a wide variety of different architectures and methods.

Algorithm	Acc (%)	ECE (%)	Speed (Sec)	w1	w2	w3	$\sum_t^A w_t = 1$
Equal-Importance	$76.8_{\pm 0.2}$	$6.3_{\pm 0.1}$	$48.3_{\pm 0.2}$	0.33	0.33	0.33	Yes
MTAN [43]	$76.5_{\pm 0.1}$	$6.5_{\pm 0.1}$	$48.5_{\pm 0.2}$	0.33	0.33	0.33	Yes
CoVV [17]	$77.3_{\pm 0.1}$	$6.5_{\pm 0.1}$	$48.9_{\pm 0.1}$	0.01	0.52	0.47	Yes
GradNorm [5]	$75.7_{\pm 0.6}$	$6.8_{\pm 0.4}$	$79.1_{\pm 0.1}$	2.99	0.00	0.00	No
MT-MOO [65]	$76.4_{\pm 0.4}$	$6.5_{\pm 0.1}$	$66.3_{\pm 0.3}$	0.36	0.47	0.16	Yes
Weighted-Importance (Ours)	$77.3_{\pm 0.4}$	$\mathbf{6.2}_{\pm 0.3}$	$48.5_{\pm 0.2}$	0.00	0.53	0.47	Yes

Table 8. Comparisons of different multi-objective optimization methods for *Peacock* evaluated on CIFAR10/CIFAR10-C using ResNet-18.

Algorithm (CIFAR10-C)	Acc (%)	ECE (%)	Speed (Sec)	w1	w2	w3	$\sum_t w_t = 1$
Equal-Importance	$82.7_{\pm 0.1}$	$9.8_{\pm 0.3}$	838 ± 3	0.33	0.33	0.33	Yes
CoVV [17]	$83.1_{\pm 0.2}$	$8.5_{\pm 0.3}$	827 ± 5	0.02	0.22	0.76	Yes
GradNorm [5]	$80.2_{\pm 0.4}$	$9.7_{\pm 0.4}$	1347 ± 3	3.00	0.00	0.00	No
MT-MOO [65]	$80.7_{\pm 0.1}$	$9.8_{\pm 0.1}$	915 ± 3	0.43	0.54	0.03	Yes
Weighted-Importance (Ours)	$81.0_{\pm 0.4}$	$\mathbf{6.1}_{\pm 0.3}$	840 ± 5	0.00	0.53	0.47	Yes

Table 9. Comparisons of different multi-objective methods for *Peacock* using SWINV2.

	(a) CIFAR10			(b) CIFAR100		
Algorithm (ID Performance)	ECE	NLL	KSE	ECE	NLL	KSE
Peacock w/o MaxE	$1.2_{\pm 0.1}$	$271.8_{\pm 2.9}$	$1.1_{\pm 0.1}$	$8.4_{\pm 0.1}$	$316.8_{\pm 1.7}$	$8.4_{\pm 0.1}$
Peacock w/o AdaFocal	$1.7_{\pm 0.1}$	289.5 ± 4.2	$1.8_{\pm 0.1}$	$6.1_{\pm 0.7}$	$314.8_{\pm 1.0}$	$6.1_{\pm 0.7}$
Peacock w/o RankMixup	$1.9_{\pm 0.2}$	$294.8_{\pm 3.2}$	$2.0_{\pm 0.2}$	$6.1_{\pm 0.3}$	$316.6_{\pm 0.9}$	$6.2_{\pm 0.3}$
Peacock w/o CPC	$1.6_{\pm 0.2}$	284.6 ± 6.3	$1.9_{\pm 0.2}$	$\mathbf{6.0}_{\pm 0.7}$	317.8 ± 1.9	$\mathbf{6.0}_{\pm 0.7}$
Peacock w/o Dual	$1.5_{\pm 0.1}$	$278.2_{\pm 2.9}$	$1.7_{\pm 0.1}$	$6.5_{\pm 0.2}$	$\mathbf{308.6}_{\pm 0.9}$	$6.5_{\pm 0.2}$
Peacock w/o ACLS	$\mathbf{0.6}_{\pm 0.1}$	$\mathbf{240.9}_{\pm 0.4}$	$\mathbf{0.9}_{\pm 0.1}$	$6.5_{\pm 0.2}$	$306.3_{\pm 0.9}$	$6.8_{\pm 0.2}$
	(a) CIFAR10-C			(b) CIFAR100-C		
Algorithm (OOD Performance)	ECE	NLL	KSE	ECE	NLL	KSE
Peacock w/o MaxE	$7.6_{\pm 0.4}$	$270.4_{\pm 2.9}$	$7.2_{\pm 0.4}$	$15.3_{\pm 0.1}$	$360.0_{\pm 0.5}$	$14.9_{\pm 0.1}$
Peacock w/o AdaFocal	$8.6_{\pm 0.4}$	$286.0_{\pm 1.3}$	$8.3_{\pm 0.4}$	$12.4_{\pm 0.6}$	$355.9_{\pm 1.3}$	$12.2_{\pm 0.6}$
Peacock w/o RankMixup	$10.1_{\pm 0.4}$	$296.8_{\pm 2.8}$	$9.9_{\pm 0.5}$	$12.7_{\pm 0.4}$	$358.4_{\pm 0.4}$	$12.4_{\pm 0.4}$
Peacock w/o CPC	$8.7_{\pm 0.4}$	$285.0_{\pm 1.8}$	$8.3_{\pm 0.3}$	$12.4_{\pm 0.7}$	$359.9_{\pm 1.7}$	$12.3_{\pm 0.7}$
Peacock w/o Dual	$8.0_{\pm 0.1}$	$278.7_{\pm 0.8}$	$7.7_{\pm 0.1}$	$12.5_{\pm 0.2}$	$\mathbf{353.2}_{\pm 1.1}$	$12.3_{\pm 0.2}$
Peacock w/o ACLS	$\mathbf{6.5}_{\pm 0.2}$	$\mathbf{245.6}_{\pm 0.4}$	$\mathbf{6.3}_{\pm 0.1}$	$\mathbf{11.6}_{\pm 0.3}$	$358.3_{\pm 0.5}$	$\mathbf{11.7}_{\pm 0.5}$

Table 10. Component analysis of *Peacock* reveals the best performance when all algorithms except ACLS are combined.

D.3. Ablation Studies

To gain a better understanding of each component in *Peacock*, we provide an ablation study that removes each component from the full combination of *Peacock*. In Tab. 10, we show the respective ECE, OE and KSE scores of each combination evaluated on CIFAR/CIFAR-C. While each component generally helps improve calibration performance, we identify RankMixup and MaxEnt loss as two of the most critical building blocks of *Peacock*. Since the removal of either RankMixup or MaxEnt loss would cause a noticeable drop in calibration performance. Although ACLS independently delivers competitive performance, we find it to be the least impactful, since its removal leads to better calibration in *Peacock*. Therefore we propose the final version of equal and importance weighted forms *Peacock* to be without ACLS. Note that the experiments performed in this ablation study does not include temperature scaling. For e.g., removing RankMixup, would cause the highest ECE on CIFAR10/CIFAR10-C with 1.9% and 10.1% respectively. The lack of MaxEnt loss constraints delivers the worst result on CIFAR100/CIFAR100-C with 8.4% and 15.3%.

E. Limitations

Component Permutations In the case of *Peacock*, we featured a total of seven baselines which gives a total of $2^7 - 1$ permutations. While the primary focus of our paper is looking at whether different calibration algorithms can be successfully combined, we constrained *Peacock* to the seven featured algorithms so as to keep experiments manageable. We note that there are many potential algorithms in the calibration family that could become promising candidates (see Fig. 2).

Modularity and Future Components To the best of our ability, we built *Peacock* based on the most relevant SOTA calibration components. For each algorithm, we closely referenced the source code provided by the respective authors. As we believe that *Peacock* will perform as well/better than

the average of its components, we specifically built *Peacock* in a modular fashion allowing the easy integration of future methods.