Pearls from Pebbles: Improved Confidence Functions for Auto-labeling

Harit Vishwakarma
hvishwakarma@cs.wisc.edu

Reid (Yi) Chen
reid.chen@wisc.edu

Sui Jiet Tay
sstay2@wisc.edu

Satya Sai Srinath Namburi
sgnamburi@cs.wisc.edu

Frederic Sala
fredsala@cs.wisc.edu

Ramya Korlakai Vinayak
ramya@ece.wisc.edu

University of Wisconsin-Madison, WI, USA

Abstract

Auto-labeling is an important family of techniques that produce labeled training sets with minimum manual labeling. A prominent variant, threshold-based auto-labeling (TBAL), works by finding a threshold on a model's confidence scores above which it can accurately label unlabeled data points. However, many models are known to produce overconfident scores, leading to poor TBAL performance. While a natural idea is to apply off-the-shelf calibration methods to alleviate the overconfidence issue, such methods still fall short. Rather than experimenting with ad-hoc choices of confidence functions, we propose a framework for studying the *optimal* TBAL confidence function. We develop a tractable version of the framework to obtain `Colander` (Confidence functions for Efficient and Reliable Auto-labeling), a new post-hoc method specifically designed to maximize performance in TBAL systems. We perform an extensive empirical evaluation of our method `Colander` and compare it against methods designed for calibration. `Colander` achieves up to 60% improvements on coverage over the baselines while maintaining auto-labeling error below 5% and using the same amount of labeled data as the baselines.

1 Introduction

The demand for labeled data in machine learning (ML) is perpetual (Fisher, 1936; Deng et al., 2009; Touvron et al., 2023), yet obtaining such data is expensive and time-consuming, creating a bottleneck in ML workflows. Threshold-based auto-labeling (TBAL) emerges as a promising solution to obtain high-quality labeled data at low cost (SGT, 2022; Qiu et al., 2023; Vishwakarma et al., 2023). A TBAL system (Figure 1) takes unlabeled data as input and outputs a labeled dataset. It works iteratively: in each iteration, it acquires human labels for a small chunk of data to train a model, then auto-labels points using the model's predictions where its *confidence scores* are above a certain threshold. The threshold is determined using validation data so that the auto-labeled points meet a desired *accuracy criteria*. The goal is to maximize *coverage*—the fraction of auto-labeled points while maintaining the accuracy criteria.

The confidence function is critical to the TBAL workflow (Figure 1). Existing TBAL systems rely on commonly used functions like softmax outputs from neural network models (Qiu et al., 2023; Vishwakarma et al., 2023). These functions *are not well aligned with the objective of the auto-labeling system*. Using them results in substantially suboptimal coverage (Figure 2(a)). Hence, a query arises:

> What are the right choices of confidence functions for TBAL and how can we obtain such functions?

An ideal confidence function for auto-labeling will achieve the maximum coverage at a given auto-labeling error tolerance and thus will bring down the labeling cost significantly. Finding such an ideal function, however, is difficult because of the *inherent tension* between accuracy and coverage. The models used in auto-labeling are often highly inaccurate so achieving a certain error guarantee is easier when being conservative in terms of confidence—but this reduces coverage. Conversely, high coverage may appear to require lowering the requirements in confidence, but this may easily lead to overshooting the error bar. This is compounded by the fact that TBAL is iterative so that even small deviations in tolerable error levels can cascade in future iterations.

Worse yet, overconfidence may ruin any hope of balancing accuracy and coverage. Furthermore, in TBAL the models are trained on a small amount of labeled data. Hence, the models are not highly accurate, making the problem of designing functions for such models even more challenging.

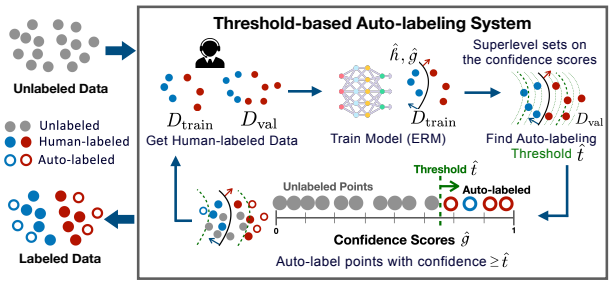


Figure 1: High-level diagram of an auto-labeling system. It takes unlabeled data as input and, with the help of expert labelers and ML models, outputs a labeled dataset.

Commonly used training procedures produce overconfident scores—high scores for both correct and incorrect predictions (Szegedy et al., 2014; Nguyen et al., 2015; Hendrycks and Gimpel, 2017; Hein et al., 2018; Bai et al., 2021). Figure 2(a) shows that the softmax scores are overconfident, resulting in poor auto-labeling performance. Several methods have been introduced to overcome overconfidence, including calibration methods (Guo et al., 2017). Using them still misses out on significant performance (Figure 2(b)) since the calibration goal differs from auto-labeling. From the auto-labeling standpoint, we want minimum overlap between the correct and incorrect model prediction scores. Other solutions (Corbière et al., 2019; Moon et al., 2020) either bake the objective of separating scores into model training or use different optimization procedures (Zhu et al., 2022) that can encourage such separation. We observe that these do not help TBAL as well since, after some point, the model is correct on almost all the training points, making it hard to train it to discriminate between its own correct and incorrect predictions.

We address these challenges by proposing a framework to learn the right confidence functions for TBAL. In particular, we express the auto-labeling objective as an optimization problem over the space of confidence functions and the thresholds. Our framework subsumes existing methods —they become points in the space of solutions. We introduce `Colander` (Confidence functions for Efficient and Reliable Auto-labeling) based on a practical surrogate to the framework that can be used to learn optimal confidence functions for auto-labeling. Using these learned functions in the TBAL can achieve up to 60% improvements in coverage versus baselines like softmax, temperature scaling (Guo et al., 2017), CRL (Moon et al., 2020) and FMFP (Zhu et al., 2022).

We summarize our contributions as follows,

1. We propose a principled framework to study the choices of confidence functions suitable for auto-labeling and provide a practical method (`Colander`) to learn confidence functions for efficient and reliable auto-labeling.
2. We systematically study commonly used choices of scoring functions and calibration methods and demonstrate that they lead to poor auto-labeling performance.
3. Through extensive empirical evaluation on real data, we show that using the confidence scores obtained using our procedure boosts auto-labeling performance significantly in comparison to common choices of confidence functions and calibration methods.

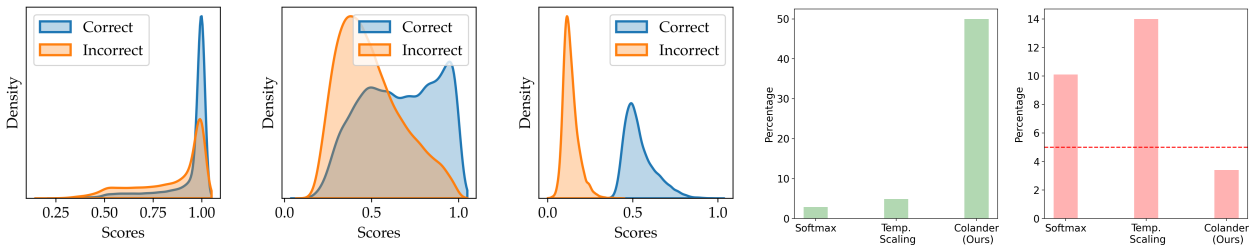
2 Background and Motivation

We begin with setting up some useful notation.

Notation. Let $[m] := \{1, 2, \ldots, m\}$ for any natural number m. Let X_u be a set of unlabeled points drawn from some instance space $\mathcal{X}$. Let $\mathcal{Y} = \{1, \ldots, k\}$ be the label space and let there be an unknown groundtruth labeling function $f^* : \mathcal{X} \to \mathcal{Y}$. Let $\mathcal{O}$ be a *noiseless* oracle that provides the true label for any point $\mathbf{x} \in \mathcal{X}$. Denote the model (hypothesis) class of classifiers by $\mathcal{H}$, where each $h \in \mathcal{H}$ is a function $h : \mathcal{X} \to \mathcal{Y}$. Each classifier h also has an associated *confidence function* $g : \mathcal{X} \to \Delta^k$ that quantifies the confidence of the prediction by model $h \in \mathcal{H}$ on any data point $\mathbf{x} \in \mathcal{X}$. Here, Δ^k is a $(k-1)$-dimensional probability simplex. Let $\mathbf{v}[i]$ denote the i^{th} component for any vector $\mathbf{v} \in \mathbb{R}^d$. For any point $\mathbf{x} \in \mathcal{X}$ the prediction is $\hat{y} := h(\mathbf{x})$ and the associated confidence is $g(\mathbf{x})[\hat{y}]$. The vector $\mathbf{t}$ denotes scores over k-classes, and $\mathbf{t}[y]$ denotes its y^{th} entry, i.e., score for class y. Please see Table 3 for a summary of the notation.

2.1 Threshold-based Auto-labeling

Threshold-based auto-labeling (TBAL) (Figure 1) seeks to obtain labeled datasets while reducing the labeling burden on humans. The input is a pool of unlabeled data X_u. It outputs, for each $\mathbf{x} \in X_u$, label $\tilde{y} \in \mathcal{Y}$. The output label could



(a) Softmax (b) Temp. Scaling (c) `Colander` (Ours) (d) Coverage (e) Auto-labeling error

Figure 2: Scores distributions (Kernel Density Estimates) of a CNN model trained on CIFAR-10 data. (a) softmax scores of vanilla training procedure (SGD) (b) scores after post-hoc calibration using temperature scaling and (c) scores from our `Colander` procedure applied on the same model. For training the CNN model we use 4000 points drawn randomly, and the number of validation points is 1000 (of which 500 are used for Temp. Scaling and `Colander`). The test accuracy of the model is 55%. Figures (d) and (e) show the coverage and auto-labeling error of these methods. The dotted-red line corresponds to a 5% error threshold.

be either y, from the oracle (representing a human-obtained label), or $\hat{y}$, from the model. Let N_u be the number of unlabeled points, $A \subseteq [N_u]$ the set of indices of auto-labeled points, and $X_u(A)$ be these points. Let N_a denote the size of the auto-labeled set A. The *auto-labeling error* denoted by $\widehat{\mathcal{E}}(X_u(A))$ and the *coverage* denoted by $\widehat{\mathcal{P}}(X_u(A))$ of the TBAL are defined as follows:

$$\widehat{\mathcal{E}}(X_u(A)) := \frac{1}{N_a} \sum_{i \in A} \mathbb{1}(\tilde{y}_i \neq f^*(\mathbf{x}_i)), \qquad (1)$$

$$\widehat{\mathcal{P}}(X_u(A)) := \frac{|A|}{N_u} = \frac{N_a}{N_u}, \qquad (2)$$

The goal of an auto-labeling algorithm is to label the dataset so that $\widehat{\mathcal{E}}(X_u(A)) \leq \epsilon_a$ while maximizing coverage $\widehat{\mathcal{P}}(X_u(A))$ for any given $\epsilon_a \in [0, 1]$. As depicted in Figure 1 the TBAL algorithm proceeds iteratively. In each iteration, it queries labels for a subset of unlabeled points from the oracle. It trains a classifier from the model class $\mathcal{H}$ on the oracle-labeled data acquired till that iteration. It then uses the model's confidence scores on the validation data to identify the region in the instance space, where the current classifier is confidently accurate and automatically labels the points in this region.

2.2 Problems with confidence functions in TBAL

The success of TBAL hinges significantly on the ability of the confidence scores of the classifier to distinguish between correct and incorrect labels. Prior works on TBAL (Vishwakarma et al., 2023; Qiu et al., 2023) train the model with Stochastic Gradient Descent (SGD) and use the softmax output of the model as confidence scores which are known to be overconfident (Nguyen et al., 2015). A natural choice to mitigate this problem is to use post-hoc calibration techniques, e.g., temperature scaling (Guo et al., 2017). We evaluate these choices by running TBAL for a single round on the CIFAR-10 (Krizhevsky et al., 2009) dataset with a SimpleCNN model with 5.8M parameters (Hussain, 2021) with error threshold 5%. See Appendix C.1 for more details.

In Figures 2(d) and 2(e) we observe that using softmax scores from the classifier only produces 2.9% coverage while the error threshold is violated with 10% error. Using temperature scaling only increases the coverage marginally to 4.9% and still violates the threshold with error 14%. Looking closer at the scores for correct versus incorrect examples on validation data, we observe a large overlap for softmax (Figure 2(a)) and a marginal shift with considerable overlap for temperature scaling (Figure 2(b)). To overcome this challenge, we propose a novel framework (Section 3) to learn such confidence functions in a principled way. Our method in this example can achieve 50% coverage with an error of 3.4% within the desired threshold (Figure 2(c)).

3 Proposed Method (`Colander`)

The observations in Figure 2(a) and 2(b) suggest that arbitrary choices of confidence functions can leave significant coverage on the table. To find a better choice of confidence function in a principled manner, we develop a framework based on auto-labeling objectives—maximizing coverage while having bounded auto-labeling error. We instantiate it by using empirical estimates and easy-to-optimize surrogates. We use the overall TBAL workflow from Vishwakarma et al. (2023) and introduce our method to replace the confidence (scoring) function after training the classifier.

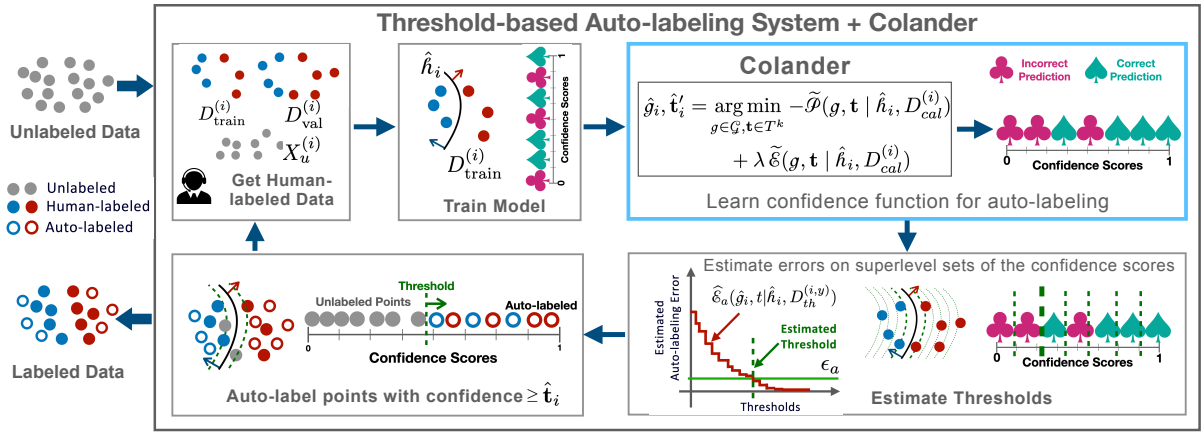


Figure 3: Threshold-based Auto-labeling with `Colander`. Similar to the existing TBAL (Figure 1) it takes unlabeled data as input, selects a small subset of data points, and obtains human labels for them to create $D^{(i)}_{\text{train}}$ and $D^{(i)}_{\text{val}}$ for the ith iteration. Then it trains model $\hat{h}_i$ on $D^{(i)}_{\text{train}}$. In contrast to the standard TBAL procedure, here we randomly split $D^{(i)}_{\text{val}}$ into two parts $D^{(i)}_{\text{cal}}$ and $D^{(i)}_{\text{th}}$. Then `Colander` kicks in, it takes $\hat{h}_i$ and $D^{(i)}_{\text{cal}}$ as input and learns coverage maximizing confidence function $\hat{g}_i$ for $\hat{h}_i$. Then using $D^{(i)}_{\text{th}}$ and $\hat{g}_i$ auto-labeling thresholds $\hat{\mathbf{t}}_i$ are determined to ensure the auto-labeled data as error at most ϵ_a. After obtaining the thresholds the rest of the steps are the same as the standard TBAL. The whole workflow runs in a loop until all the data is labeled or some other stopping criteria are achieved.

3.1 Auto-labeling optimization framework

In any iteration of TBAL, we have a model h trained on a subset of data labeled by the oracle. This model may not be highly accurate. However, it could be accurate in some regions of the instance space, and with the help of a confidence function g, we want to identify the points where the model is correct and auto-label them. As we saw earlier, arbitrary choices of g perform poorly on this task. Instead of relying on these choices, we propose a framework to find the right function from a sufficiently rich family of confidence functions that also subsumes the current choices.

Optimal confidence function. To find the confidence function aligned with our objective, we consider a rich enough space of the confidence functions $\mathcal{G}$ and thresholds and express the auto-labeling objective as an optimization problem (P1) over these spaces.

$$\underset{g\in\mathcal{G},\mathbf{t}\in T^k}{\arg\max} \quad \mathcal{P}(g,\mathbf{t}\mid h) \;\text{ s.t. }\; \mathcal{E}(g,\mathbf{t}\mid h)\le\epsilon_a. \tag{P1}$$

Here T is the set of confidence thresholds and $\mathcal{G}:\mathcal{X}\to T^k$ is the set of confidence functions and $\mathcal{P}(g,\mathbf{t}|h)$ and $\mathcal{E}(g,\mathbf{t}\mid h)$ are the population level coverage and auto-labeling error which are defined as follows,

$$\mathcal{P}(g,\mathbf{t}\mid h) := \mathbb{P}_{\mathbf{x}}\big(g(\mathbf{x})[\hat{y}]\ge\mathbf{t}[\hat{y}]\big), \tag{3}$$

$$\mathcal{E}(g,\mathbf{t}\mid h) := \mathbb{P}_{\mathbf{x}}\big(y\neq\hat{y}\mid g(\mathbf{x})[\hat{y}]\ge\mathbf{t}[\hat{y}]\big). \tag{4}$$

The optimal $g^\star$ and $\mathbf{t}^\star$ that achieve the maximum coverage while satisfying the auto-labeling error constraint belong to the solution(s) of the following optimization problem.

3.2 Practical method to learn confidence functions

The above framework provides a theoretical characterization of the optimal confidence functions and thresholds for TBAL. However, it is not practical since, in practice, the data distributions and $f^\star$ are not known. Next, we provide a practical method based on the above framework to learn confidence functions for TBAL.

Empirical optimization problem. Since we do not know the distributions of $\mathbf{x}$ and $f^\star$, we use estimates of coverage and auto-labeling errors on a fraction of validation data to solve the optimization problem. Let D be some finite number of labeled samples, and then the empirical coverage and auto-labeling error are defined as follows,

$$\widehat{\mathcal{P}}(g, \mathbf{t} \mid \hbar, D) := \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \mathbb{1}\big(g(\mathbf{x})[\hat{y}] \geq \mathbf{t}[\hat{y}]\big), \tag{5}$$

$$\widehat{\mathcal{E}}(g, \mathbf{t} \mid \hbar, D) := \frac{\sum_{(\mathbf{x},y)\in D} \mathbb{1}\big(y \neq \hat{y} \wedge g(\mathbf{x})[\hat{y}] \geq \mathbf{t}[\hat{y}]\big)}{\sum_{(\mathbf{x},y)\in D} \mathbb{1}\big(g(\mathbf{x})[\hat{y}] \geq \mathbf{t}[\hat{y}]\big)}. \tag{6}$$

We randomly split the validation data into two parts D_{cal} and D_{th} and use D_{cal} to compute $\widehat{\mathcal{P}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}})$ and $\widehat{\mathcal{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}})$ for the following empirical version of the optimization problem. We now hope to solve the following optimization problem using these estimates to get $\hat{g}, \hat{\mathbf{t}}$.

$$\underset{g\in\mathcal{G}, \mathbf{t}\in T^k}{\arg\max} \quad \widehat{\mathcal{P}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) \;\text{ s.t. }\; \widehat{\mathcal{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) \leq \epsilon_a. \tag{P2}$$

However, there is a caveat: the objective and constraint are based on 0-1 variables, so it is hard to optimize for g and $\mathbf{t}$.

Surrogate optimization problem. To make the above optimization (P2) tractable using gradient-based methods, we introduce differentiable surrogates for the 0-1 variables. Let $\sigma(\alpha, z) := 1/(1 + \exp(-\alpha z))$ denote the sigmoid function on $\mathbb{R}$ with scale parameter $\alpha \in \mathbb{R}$. It is easy to see that, for any g, y and $\mathbf{t}$, $g(\mathbf{x})[y] \geq \mathbf{t}[y] \iff \sigma(\alpha, g(\mathbf{x})[y] - \mathbf{t}[y]) \geq 1/2$. Using this fact, we define the following surrogates of the auto-labeling error and coverage as follows,

$$\widetilde{\mathcal{P}}(g, \mathbf{t} | \hbar, D_{\text{cal}}) := \frac{1}{|D_{\text{cal}}|} \sum_{(\mathbf{x},y)\in D_{\text{cal}}} \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big), \tag{7}$$

$$\widetilde{\mathcal{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) := \frac{\sum_{(\mathbf{x},y)\in D_{\text{cal}}} \mathbb{1}\big(y \neq \hat{y}\big)\, \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big)}{\sum_{(\mathbf{x},y)\in D_{\text{cal}}} \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big)}. \tag{8}$$

and the surrogate optimization problem as follows,

$$\underset{g\in\mathcal{G}, \mathbf{t}\in T^k}{\arg\min} \quad -\widetilde{\mathcal{P}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) + \lambda\, \widetilde{\mathcal{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) \tag{P3}$$

Here, $\lambda \in \mathbb{R}^+$ is the penalty term controlling the relative importance of the auto-labeling error and coverage. It is a hyper-parameter, and we find it using our hyper-parameter searching procedure discussed in section 4.3. The gap between the surrogate and actual coverage, error diminishes as $\alpha \to \infty$. We discuss this in Appendix B.

Choice of $\mathcal{G}$. Our framework is flexible with the function class $\mathcal{G}$ choice. In this work, we use deep neural networks (DNNs) with at least two layers on model class $\mathcal{H}$. Since DNNs also learn powerful representations during training, we use the last two layers of representations as input for the functions in $\mathcal{G}$ (Figure 4). Let $\mathbf{z}^{(1)}(\mathbf{x}; \hbar) \in \mathbb{R}^k$ and $\mathbf{z}^{(2)}(\mathbf{x}; h) \in \mathbb{R}^{d_2}$ be the outputs(logits) of the last and the second-last layer of the net h for input $\mathbf{x}$ and let $\mathbf{z}(\mathbf{x}; h) := [\mathbf{z}^{(1)}(\mathbf{x}; \hbar), \mathbf{z}^{(2)}(\mathbf{x}; \hbar)]$ denote the concatenation of the two representations. We propose to use two-layer neural networks $\mathcal{G}_{nn_2} : \mathbb{R}^{k+d_2} \mapsto \Delta^k$ for $\mathcal{G}$. A net $g \in \mathcal{G}_{nn_2}$ takes the last two layer's representations from h and outputs confidence scores over k classes. Given $\hbar$, the g is defined as follows,

$$g(\mathbf{x}) := \texttt{softmax}\big(\mathbf{W}_2 \texttt{tanh}(\mathbf{W}_1 \mathbf{z}(\mathbf{x}; \hbar))\big). \tag{9}$$

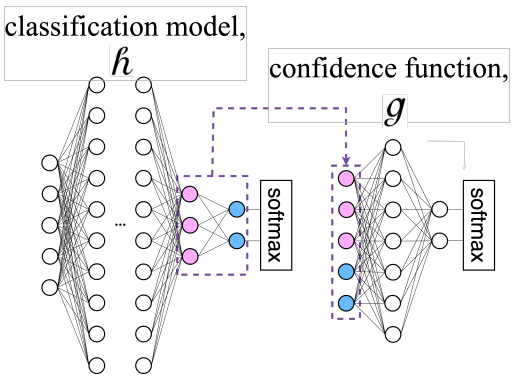


Figure 4: Our choice of g function.

Here $\mathbf{W}_1 \in \mathbb{R}^{(k+d_2)\times 2(k+d_2)}$ and $\mathbb{R}^{2(k+d_2)\times k}$ are the learnable weight matrices and for any $\mathbf{v} \in \mathbb{R}^d$, the $\texttt{softmax}(\mathbf{v})[i] := \exp(\mathbf{v}[i])/(\sum_j \exp(\mathbf{v}[j]))$ and $\texttt{tanh}(\mathbf{v})[i] := (\exp(2\mathbf{v}[i]) - 1)/(\exp(2\mathbf{v}[i]) + 1)$.

Algorithm 1 Threshold-based Auto-Labeling (TBAL)

Input: Unlabeled data X_u, labeled validation data D_{val}, auto labeling error tolerance ϵ_a, N_t training data query budget, seed data size n_s, batch size for active query n_b, calibration data fraction ν, set of confidence thresholds T, coverage lower bound ρ_0, label space $\mathcal{Y}$.

Output: Auto-labeled dataset D_{out}

procedure TBAL($X_u, D_{\text{val}}, \epsilon_a, N_t, n_s, n_b, \nu, \rho_0, T, \mathcal{Y}$)
  ▷ /*** Initialization. ***/
  $D^{(1)}_{\text{query}} \leftarrow$ RANDOMQUERY(X_u, n_s) ▷ Randomly select n_s points and get manual labels for them.
  $X^{(1)}_u \leftarrow X_u \setminus \{\mathbf{x} : (\mathbf{x}, y) \in D^{(1)}_{\text{query}}\}$ ▷Remove the manually labeled points from the unlabeled pool.
  $D^{(1)}_{\text{val}} \leftarrow D_{\text{val}}; D^{(0)}_{\text{train}} \leftarrow \emptyset$ ▷Validation data for the first round is full D_{val}.
  $D_{\text{out}} \leftarrow D^{(1)}_{\text{query}}; n^{(1)}_t \leftarrow n_s; i \leftarrow 1$ ▷Include the manually labeled data in Step 2. in the output data D_{out}.
  ▷ /*** Run the auto-labeling loop ***/
  ▷ /* Until no more unlabeled points are left or the budget for manually labeled training data is exhausted. */
  while $X^{(i)}_u \neq \emptyset$ and $n^{(i)}_t \leq N_t$ **do**
    $D^{(i)}_{\text{train}} \leftarrow D^{(i-1)}_{\text{train}} \cup D^{(i)}_{\text{query}}$ ▷Include the manually labeled points in the training data.
    $\hat{h}_i \leftarrow$ TRAINMODEL$(\mathcal{H}, D^{(i)}_{\text{train}})$ ▷Train a classification model.
    $D^{(i)}_{\text{cal}}, D^{(i)}_{\text{th}} \leftarrow$ RANDOMSPLIT$(D^{(i)}_{\text{val}}, \nu)$ ▷Randomly split the current validation data into two parts.
    ▷ /*** `Colander` block, to learn the new confidence function $\hat{g}_i$***/
    $\hat{g}_i, \hat{\mathbf{t}}'_i \leftarrow \arg\min_{g \in \mathcal{G}, \mathbf{t} \in T^k} -\widetilde{\mathcal{P}}(g, \mathbf{t} \mid \hat{h}_i, D^{(i)}_{\text{cal}}) + \lambda \widetilde{\mathcal{E}}(g, \mathbf{t} \mid \hat{h}_i, D^{(i)}_{\text{cal}})$ ▷ `Colander` procedure.
    ▷ /*** Estimate auto-labeling thresholds using $\hat{g}_i$ and $D^{(i)}_{\text{th}}$. See Algorithm 2. ***/
    $\hat{\mathbf{t}}_i \leftarrow$ ESTTHRESHOLD$(\hat{g}_i, \hat{h}_i, D^{(i)}_{\text{th}}, \epsilon_a, \rho_0, T, \mathcal{Y})$
    ▷ /*** Auto-label the points having scores above the thresholds. ***/
    $\widetilde{D}^{(i)}_u \leftarrow \{(\mathbf{x}, \hat{h}_i(\mathbf{x})) : \mathbf{x} \in X^{(i)}_u\}$
    $D^{(i)}_{\text{auto}} \leftarrow \{(\mathbf{x}, \hat{y}) \in \widetilde{D}^{(i)}_u : \hat{g}_i(\mathbf{x})[\hat{y}] \geq \hat{\mathbf{t}}_i[\hat{y}]\}$
    $X^{(i)}_u \leftarrow X^{(i)}_u \setminus \{\mathbf{x} : (\mathbf{x}, \hat{y}) \in D^{(i)}_{\text{auto}}\}$ ▷Remove auto-labeled points from the unlabeled pool.
    $\widetilde{D}^{(i)}_{\text{val}} \leftarrow \{(\mathbf{x}, \hat{h}_i(\mathbf{x})) : (\mathbf{x}, y) \in D^{(i)}_{\text{val}}\}$
    $D^{(i+1)}_{\text{val}} \leftarrow \{(\mathbf{x}, \hat{y}) \in \widetilde{D}^{(i)}_{\text{val}} : \hat{g}_i(\mathbf{x})[\hat{y}] < \hat{\mathbf{t}}_i[\hat{y}]\}$ ▷Remove validation points in the auto-labeling region.
    ▷ /*** Get the next batch of manually labeled data using an active querying strategy. ***/
    $D^{(i+1)}_{\text{query}} \leftarrow$ ACTIVEQUERY$(\hat{h}_i, X^{(i)}_u, n_b)$
    $X^{(i+1)}_u \leftarrow X^{(i)}_u \setminus \{\mathbf{x} : (\mathbf{x}, y) \in D^{(i+1)}_{\text{query}}\}$ ▷Remove manually labeled data from the unlabeled pool.
    $D_{\text{out}} \leftarrow D_{\text{out}} \cup D^{(i)}_{\text{auto}} \cup D^{(i+1)}_{\text{query}}$ ▷Add the auto-labeled and manually labeled points in the output data.
    $n^{(i+1)}_t \leftarrow n^{(i)}_t + n_b$
    $i \leftarrow i + 1$
  end while
  return D_{out}
end procedure

Solving the surrogate optimization. The optimization problem (P3) is non-convex even for a simple class of g (such as linear). Nevertheless, it is differentiable and we apply gradient-based methods, which have been highly effective in minimizing non-convex losses in deep learning. We solve for g and $\mathbf{t}$ simultaneously using the Adam optimizer (Kingma and Ba, 2014). The details of the training hyperparameters are deferred to the Appendix C.4.

3.3 TBAL procedure with `Colander`

We plugin our method `Colander` to learn confidence functions in the workflow of TBAL (Algorithm 1). The workflow is also illustrated in Figure 3. The steps in the updated workflow are the same as the standard TBAL (Figure 1), except for the introduction of `Colander` after the model training step to learn a new confidence function $\hat{g}_i$ using part of

Algorithm 2 Estimate Auto-Labeling Threshold

Input: Confidence function $\hat{g}_i$, classifier $\hat{h}_i$, Part of validation data $D_{\text{th}}^{(i)}$ for threshold estimation, auto labeling error tolerance ϵ_a, set of confidence thresholds T, coverage lower bound ρ_0, label space $\mathcal{Y}$.
Output: Auto-labeling thresholds $\hat{\mathbf{t}}_i$, where $\hat{\mathbf{t}}_i[y]$ is the threshold for class y.

procedure EstThreshold($\hat{g}_i, \hat{h}_i, D_{\text{th}}^{(i)}, \epsilon_a, \rho_0, T, \mathcal{Y}$)
  ▷ /*** Estimate thresholds for each class. ***/
  for $y \in \mathcal{Y}$ **do**
    $D_{\text{th}}^{(i,y)} \leftarrow \{(\mathbf{x}', y') \in D_{\text{th}}^{(i)} : y' = y\}$ ▷Group points class-wise.
    ▷ /*** Only evaluate thresholds with est. coverage at least ρ_0. ***/
    $T'_y \leftarrow \{t \in T : \widehat{\mathscr{P}}(\hat{g}_i, t \mid \hat{h}_i, D_{\text{th}}^{(i,y)}) \geq \rho_0\} \cup \{\infty\}$
    ▷ /*** Estimate auto-labeling error at each threshold. Pick the smallest threshold with the sum of estimated error and C_1 times the standard deviation is below ϵ_a. C_1 is set to 0.25 here. ***/
    $\hat{\mathbf{t}}_i[y] \leftarrow \min\{t \in T'_y : \widehat{\mathcal{E}}_a(\hat{g}_i, t|\hat{h}_i, D_{\text{th}}^{(i,y)}) + C_1\hat{\sigma}(\hat{h}_i, t, D_{\text{th}}^{(i,y)}) \leq \epsilon_a\}$
  end for
  return $\hat{\mathbf{t}}_i$
end procedure

the validation data ($D_{\text{cal}}^{(i)}$) and then the threshold estimation procedure (Algorithm 2) finds auto-labeling thresholds $\hat{\mathbf{t}}_i$ on the scores computed using $\hat{g}_i$ on the other part of the validation data called $D_{\text{th}}^{(i)}$. While we get thresholds as output from `Colander`, it is important to estimate them again from the held-out data $D_{\text{th}}^{(i)}$ to ensure the auto-labeling error constraint is not violated. In Algorithm 1 the procedure RandomQuery(X_u, n_s) selects n_s points randomly from X_u and obtains human labels for them to create $D_{\text{train}}^{(1)}$. The procedure RandomSplit$(D_{\text{val}}^{(i)}, \nu)$ randomly splits $D_{\text{val}}^{(i)}$, the validation data in i^{th} iteration to $D_{\text{cal}}^{(i)}$ and $D_{\text{th}}^{(i)}$ with $D_{\text{cal}}^{(i)}$ having ν fraction of points from $D_{\text{val}}^{(i)}$. $D_{\text{cal}}^{(i)}$ is used for learning the post-hoc confidence function and $D_{\text{th}}^{(i)}$ is used for estimating auto-labeling thresholds in Algorithm 2. The procedure, TrainModel$(\mathcal{H}, D_{\text{train}}^{(i)})$ trains a model from model class $\mathcal{H}$ on the training data $D_{\text{train}}^{(i)}$. Any training procedure can be used here, in this work we use methods listed in Section 4.1.1 for model training. Lastly, ActiveQuery$(\hat{h}_i, X_u^{(i)}, n_b)$, selects n_b points from the remaining unlabeled pool using the same active learning strategy used in a prior work (Vishwakarma et al., 2023). We defer the details to the Appendix B.

4 Empirical Evaluation

As we observed in Section 2.2, ad-hoc choices of confidence functions can lead to poor auto-labeling performance. Motivated by these shortcomings, we designed a method to learn confidence functions that are well-aligned with the auto-labeling objective. In this section, we verify the following claims through extensive empirical evaluation,

C1. `Colander` learns better confidence functions for auto-labeling compared to standard training and common post-hoc methods that mitigate the overconfidence problem. Using it in TBAL can boost the coverage significantly while keeping the auto-labeling error low.

C2. `Colander` is not dependent on any particular train-time method and thus should improve the performance over using any train-time method alone.

4.1 Baselines

We compare several train-time and post-hoc methods that improve confidence functions from calibration and ordinal ranking perspectives. Detailed descriptions of these methods are deferred to Appendix C.5.

4.1.1 Train-time methods

We use the following methods for training the model $\hat{h}$.

1. *Vanilla* neural network trained under cross-entropy loss using stochastic gradient descent (SGD) (Amari, 1993; Bottou, 2012; Guo et al., 2017).

Dataset	Model $\hbar$	N	N_u	K	N_t	N_v	N_{hyp}	Modality	Preprocess	Dimension
MNIST	LeNet-5	70k	60k	10	500	500	500	Image	None	$1 \times 28 \times 28$
CIFAR-10	CNN	50k	40k	10	10k	8k	2k	Image	None	$3 \times 32 \times 32$
Tiny-Imagenet	MLP	110k	90k	200	10k	8k	2k	Image	CLIP	512
20 Newsgroup	MLP	11.3k	9k	20	2k	1.6k	600	Text	FlagEmb.	1,024

Table 1: Details of the dataset and model we used to evaluate the performance of our method and other calibration methods. For the Tiny-Imagenet and 20 Newsgroup datasets, we use CLIP and FlagEmbedding, respectively, to obtain the embeddings of these datasets and conduct auto-labeling on the embedding space. For Tiny-Imagenet, we use a 3-layer perceptron with 1,000, 500, 300 neurons on each layer as model $\hbar$; for 20 Newsgroup, we use a 3-layer perceptron with 1,000, 500, 30 neurons on each layer as model $\hbar$.

2. *Squentropy* (Hui et al., 2023) adds the average square loss over the incorrect classes to cross-entropy loss to improve the calibration and accuracy of the model.
3. *Correctness Ranking Loss (CRL)* (Moon et al., 2020) aligns the confidence scores of the model with the ordinal rankings criterion via regularization.
4. *FMFP* (Zhu et al., 2022) aligns confidence scores with the ordinal rankings criterion by using Sharpness Aware Minimization (SAM) (Foret et al., 2021) in lieu of SGD.

4.1.2 Post-hoc methods

We use the following methods for learning (or updating) the confidence function $\hat{g}$ after learning $\hat{\hbar}$.

1. *Temperature scaling* (Guo et al., 2017) is a variant of Platt scaling (Platt, 1999). It rescales the logits by a learnable scalar parameter.
2. *Top-Label Histogram-Binning* (Gupta and Ramdas, 2022) builds on the histogram-binning method (Zadrozny and Elkan, 2002) and focuses on calibrating the scores of the predicted label assigned to unlabeled points.
3. *Scaling-Binning* (Kumar et al., 2019) applies temperature scaling and then bins the confidence function values.
4. *Dirichlet Calibration* (Kull et al., 2019) models the distribution of predicted probability vectors separately on instances of each class and assumes Dirichlet class conditional distributions.

Remark: Each train-time method is piped with a post-hoc method, yielding total $4 \times 5 = 20$ methods.

4.2 Datasets and models

We evaluate the performance of auto-labeling on four datasets. Each is paired with a model for auto-labeling:

1. *MNIST* LeCun (1998) is a hand-written digits dataset. We use the LeNet LeCun et al. (1998) for auto-labeling.
2. *CIFAR-10* Krizhevsky et al. (2009) is an image dataset with 10 classes. We use a CNN with approximately 5.8M parameters Hussain (2021) for auto-labeling.
3. *Tiny-ImageNet* Le and Yang (2015) is an image dataset comprising 100K images across 200 classes. We use CLIP Radford et al. (2021) to derive embeddings for the images in the dataset and use an MLP model.
4. *20 Newsgroups* Mitchell (1999) is a natural language dataset comprising around 18K news posts across 20 topics. We use the FlagEmbedding Xiao et al. (2023) to obtain text embeddings and use an MLP model.

4.3 Hyperparameter Search and Evaluation

The complexity of TBAL workflow and lack of labeled data make hyperparameter search and evaluation challenging. Similar challenges have been observed in active learning (Lowell et al., 2019). We discuss our practical approach and defer the details to Appendix C.8.

Hyperparameter Search. We run only the first round of TBAL with each method using a hyperparameter combination 5 times and measure the mean auto-labeling error and mean coverage on D_{hyp}, which represents a small part of the held-out human-labeled data. We pick the combination that yields the lowest average auto-labeling error while maximizing the coverage. We first find the best hyperparameters for each train-time method, fix those, and then search the hyperparameters for the post-hoc methods. Note that the best hyperparameter for a post-hoc method depends on the

Train-time	Post-hoc	MNIST		CIFAR-10		20 Newsgroups		Tiny-ImageNet	
		Err (↓)	Cov (↑)	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)
Vanilla	Softmax	**4.1**±0.7	85.0±2.5	4.8±0.2	14.0±2.1	6.0±0.6	48.2±1.6	11.1±0.3	32.6±0.5
	TS	7.8±0.6	94.2±0.5	7.3±0.3	23.2±0.7	9.7±0.6	60.7±2.3	16.3±0.5	37.4±1.5
	Dirichlet	7.9±0.7	93.2±2.2	7.7±0.5	22.4±1.2	9.4±0.9	59.4±1.8	17.1±0.4	33.3±2.0
	SB	6.7±0.5	92.6±1.5	6.1±0.4	18.6±1.1	8.1±0.6	58.1±1.8	15.7±0.6	35.4±1.2
	Top-HB	7.4±1.4	93.1±3.6	6.0±0.7	15.6±1.9	9.2±1.0	59.0±2.0	16.6±0.5	37.6±2.2
	Ours	4.2±1.5	**95.6**±1.4	**3.0**±0.2	**78.5**±0.2	**2.5**±1.1	**80.6**±0.7	**1.4**±2.1	**59.2**±0.8
CRL	Softmax	4.7±0.4	86.0±4.5	5.2±0.3	15.9±0.8	5.8±0.5	48.3±0.3	10.4±0.4	32.5±0.6
	TS	8.0±0.8	94.8±0.8	6.8±0.8	20.3±1.1	9.5±1.0	61.7±1.6	15.8±0.6	37.4±1.7
	Dirichlet	8.6±0.6	93.1±1.6	7.7±0.2	20.9±1.1	8.7±0.9	58.0±1.4	16.3±0.4	33.1±1.9
	SB	7.4±0.8	93.1±2.7	5.9±0.9	17.9±1.5	8.9±1.1	57.9±3.9	15.0±0.4	35.5±1.2
	Top-HB	7.7±0.8	94.1±1.5	4.4±0.5	12.3±0.4	8.8±1.0	58.8±2.7	16.5±0.5	38.9±1.6
	Ours	**4.5**±1.4	**95.6**±1.3	**2.2**±0.6	**77.9**±0.2	**1.8**±1.2	**81.3**±0.5	**2.8**±2.1	**61.2**±1.4
FMFP	Softmax	4.8±0.8	84.2±4.1	4.9±0.4	15.6±1.7	5.4±0.7	45.4±1.9	10.5±0.3	32.4±1.4
	TS	8.0±0.6	95.3±1.6	6.5±0.3	21.0±1.5	9.5±0.5	57.7±2.2	16.2±1.1	37.7±1.8
	Dirichlet	8.2±1.3	94.0±2.2	6.9±0.4	21.7±1.2	8.9±1.0	56.6±2.4	17.4±0.8	33.0±1.8
	SB	7.2±1.1	93.1±2.3	6.1±0.5	19.5±1.0	8.6±0.4	55.8±1.3	15.5±0.6	36.1±0.5
	Top-HB	7.1±0.6	93.3±4.9	5.2±0.5	14.2±2.4	9.0±0.7	57.9±2.4	16.2±0.4	37.4±1.1
	Ours	**4.6**±0.8	**95.7**±0.2	**3.0**±0.4	**77.4**±0.2	**2.5**±0.9	**80.8**±0.6	**1.8**±2.0	**60.8**±1.4
Squentropy	Softmax	**3.7**±1.0	88.2±3.9	5.2±0.5	21.2±1.8	4.6±0.4	52.0±1.2	7.8±0.3	36.2±0.8
	TS	6.2±1.1	95.6±0.9	6.9±0.6	28.2±2.5	8.3±0.6	66.6±1.4	13.3±0.1	44.9±1.0
	Dirichlet	6.5±1.2	95.9±0.8	7.3±0.3	29.4±1.1	7.8±0.6	64.0±1.3	14.1±0.3	42.5±0.7
	SB	6.0±0.8	95.3±1.2	6.2±0.4	23.8±1.9	7.8±0.7	63.0±2.9	13.0±0.5	45.2±2.0
	Top-HB	5.3±0.4	96.4±0.9	4.3±0.5	15.8±1.4	8.2±0.8	66.5±2.2	13.7±0.1	45.9±1.4
	Ours	4.1±0.8	**97.2**±0.5	**2.3**±0.5	**79.0**±0.3	**3.3**±0.8	**82.9**±0.4	**0.6**±0.2	**66.5**±0.7

Table 2: In every round the error was enforced to be below 5%; 'TS' stands for Temperature Scaling, 'SB' stands for Scaling Binning, 'Top-HB' stands for Top-Label Histogram Binning. The column Err stands for auto-labeling error and Cov stands for the coverage. Each cell value is mean ± std. deviation observed on 5 repeated runs with different random seeds.

training-time method that it pipes to. The hyperparameter search spaces are in the Appendix C; and the selected values used for each setting are in the supplementary material.

Performance Evaluation. After fixing the hyper-parameters, we run TBAL with each combination of train-time and post-hoc method on full X_u of size N, with a fixed budget of N_t labeled training samples and N_v validation samples. The details of these values for each dataset are in Table 1 in Appendix C. Here, we know the ground truth labels for the points in X_u, so we measure the auto-labeling error and coverage as defined in equations (1) and (2) respectively and report them in Table 2. We discuss these results and their implications in the next section.

4.4 Results and Discussion

Our findings are, shown in Table 2, are:

C1: `Colander` ***improves TBAL performance.*** Our approach aims to optimize the confidence function to maximize coverage while minimizing errors. When applied to TBAL, we expect it to yield substantial coverage enhancement and error reduction compared to vanilla training and softmax scores. Indeed, the results in Table 2 corresponding

to the vanilla training match our expectations. We see *across all data settings*, our method achieves *significantly higher coverage* while keeping auto-labeling error below the tolerance level of 5%. The improvements are even more pronounced when the datasets are more complex than MNIST. Also consistent with our expectation and observations in Figure 2(b), the post-hoc calibration methods improve the coverage over using softmax scores but at the cost of slightly higher error. While they are reasonable choices to apply in the TBAL pipeline, they fall short of maximally improving TBAL performance due to the misalignment of goals.

C2: `Colander` is compatible with and improves over other train-time methods. Our method is compatible with various choices of train-time methods, and if a train-time method (Squentropy here) provides a *better* model relative to another train-time method (e.g., Vanilla), then our method exploits this gain and pushes the performance even further. Across different train-time methods, we do not see significant differences in the performance, except for Squentropy. Using Squentropy with softmax improves the coverage by as high as 6-7% while dropping the auto-labeling error in contrast to using softmax scores obtained with other train-time methods for the Tiny-ImageNet setting. This is an unexpected and interesting finding. Squentropy adds average square loss over the incorrect classes as a regularizer, and it has been shown to achieve better accuracy and calibration compared to training just with cross-entropy loss (Vanilla).

Train-time methods designed for ordinal ranking objective perform poorly in auto-labeling. CRL and FMFP are state-of-the-art methods designed to produce scores aligned with the ordinal ranking criteria. Ideally, if the scores satisfy this criterion, TBAL's performance would improve. However, we do not see any significant difference from the Vanilla method. Similar to the other baselines, their evaluation is focused on models trained on large amounts of data. But, in TBAL, we have less data for training. The training error goes to zero after some rounds, and no information is left for the CRL loss to distinguish between correct and incorrect predictions (i.e., count SGD mistakes). On the other hand, FMFP is based on a hypothesis that training models using Sharpness Aware Minimizer (SAM) could lead to scores satisfying the ordinal ranking criteria. However, this phenomenon is still not well understood, especially in settings like ours with limited training data.

5 Related Works

Data-labeling. We briefly discuss prominent methods for data labeling. Crowdsourcing Raykar et al. (2010); Sorokin and Forsyth (2008) uses a crowd of non-experts to complete a set of labeling tasks. Works in this domain focus on mitigating noise in the obtained information, modeling label errors, and designing effective labeling tasks Gomes et al. (2011); Karger et al. (2011); Mazumdar and Saha (2017); Vinayak et al. (2014); Vinayak and Hassibi (2016); Vinayak et al. (2017); Chen et al. (2023). Weak supervision, in contrast, emphasizes labeling through multiple inexpensive but noisy sources, not necessarily human (Ratner et al., 2016; Fu et al., 2020; Shin et al., 2022; Vishwakarma and Sala, 2022). Works such as Ratner et al. (2016); Fu et al. (2020) concentrate on binary or multi-class labeling, while Shin et al. (2022); Vishwakarma and Sala (2022) extend weak supervision to structured prediction tasks.

Auto-labeling occupies an intermediate position between weak supervision and crowdsourcing in terms of human dependency. It aims to minimize costs associated with obtaining labels from humans while generating high-quality labeled data using a specific machine learning model. Qiu et al. (2023) use a TBAL-like algorithm and explore the cost of training for auto-labeling with large-scale model classes. Recent work Vishwakarma et al. (2023) theoretically analyzes the sample complexity of validation data required to guarantee the quality of auto-labeled data.

Overconfidence and calibration. The issue of overconfidence (Szegedy et al., 2014; Nguyen et al., 2015; Hein et al., 2018; Bai et al., 2021) is detrimental in several applications, including ours. Many solutions have emerged to mitigate the overconfidence and miscalibration problem. Gawlikowski et al. (2021) provide a comprehensive survey on uncertainty quantification and calibration techniques for neural networks. Guo et al. (2017) evaluated a variety of solutions ranging from the choice of network architecture, model capacity, weight decay regularization (Krogh and Hertz, 1991), histogram-binning and isotonic regression (Zadrozny and Elkan, 2001, 2002) and temperature scaling (Platt, 1999; Niculescu-Mizil and Caruana, 2005) which they found to be the most promising solution. The solutions fall into two broad categories: train-time and post-hoc. Train-time solutions modify the loss function, include additional regularization terms, or use different training procedures (Kumar et al., 2018; Müller et al., 2019; Mukhoti et al., 2020; Hui et al., 2023). On the other hand, post-hoc methods such as top-label histogram-binning (Gupta and Ramdas, 2021), scaling binning (Kumar et al., 2019), Dirichlet calibration (Kull et al., 2019) calibrate the scores directly or learn a model that corrects miscalibrated confidence scores.

Beyond calibration. While calibration aims to match the confidence scores with a probability of correctness, it is not the precise solution to the overconfidence problem in many applications, including our setting. The desirable criteria for scores for TBAL are closely related to the ordinal ranking criterion (Hendrycks and Gimpel, 2017). To get such scores, Corbière et al. (2019) add an additional module in the net for failure prediction, Zhu et al. (2022) switch to sharpness aware minimization Foret et al. (2021) to learn the model; CRL (Moon et al., 2020) regularizes the loss function.

6 Conclusions

We studied issues with confidence scoring functions used in threshold-based auto-labeling (TBAL). We showed that the commonly used confidence functions and calibration methods can often be a bottleneck, leading to poor performance. We proposed `Colander` to learn confidence functions that are aligned with the TBAL objective. We evaluated our method extensively against common baselines on several real-world datasets and found that it improves the performance of TBAL significantly in comparison to the several common choices of confidence function. Our method is compatible with several choices of methods used for training the classifier in TBAL and using it in conjunction with them improves TBAL performance further. A limitation of `Colander` is that, similar to other post-hoc methods it also requires validation data to learn the confidence function. Reducing (or eliminating) this dependence on validation data could be an interesting future work.

7 Acknowledgments

This work was partly supported by funding from the American Family Data Science Institute. We thank Heguang Lin, Changho Shin, Dyah Adila, Tzu-Heng Huang, John Cooper, Aniket Rege, Daiwei Chen and Albert Ge for their valuable inputs. We thank the anonymous reviewers for their valuable comments and constructive feedback on our work.

References

S.-i. Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

Y. Bai, S. Mei, H. Wang, and C. Xiong. Don't just blame over-parametrization for over-confidence: Theoretical analysis of calibration in binary classification. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 566–576. PMLR, 18–24 Jul 2021.

L. Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35289-8.

Y. Chen, R. K. Vinayak, and B. Hassibi. Crowdsourced clustering via active querying: Practical algorithm with theoretical guarantees. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 11, pages 27–37, 2023.

C. Corbière, N. THOME, A. Bar-Hen, M. Cord, and P. Pérez. Addressing failure prediction by learning model confidence. In *Advances in Neural Information Processing Systems 32*, pages 2902–2913. 2019.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugenics*, 7:179–188, 1936.

P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

D. Y. Fu, M. F. Chen, F. Sala, S. M. Hooper, K. Fatahalian, and C. Ré. Fast and three-rious: Speeding up weak supervision with triplet methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, 2020.

J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

R. G. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *Advances in Neural Information Processing Systems 24*, pages 558–566. 2011.

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

C. Gupta and A. Ramdas. Distribution-free calibration guarantees for histogram binning without sample splitting. In *International Conference on Machine Learning*, pages 3942–3952. PMLR, 2021.

C. Gupta and A. Ramdas. Top-label calibration and multiclass-to-binary reductions. In *International Conference on Learning Representations*, 2022.

S. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

M. Hein, M. Andriushchenko, and J. Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. 2018.

D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017.

L. Hui, M. Belkin, and S. Wright. Cut your losses with squentropy. In *Proceedings of the 40th International Conference on Machine Learning*, pages 14114–14131, 2023.

S. Hussain. Cifar 10- cnn using pytorch, Jul 2021. URL https://www.kaggle.com/code/shadabhussain/cifar-10-cnn-using-pytorch.

D. R. Karger, S. Oh, and D. Shah. Budget-optimal crowdsourcing using low-rank matrix approximations. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 284–291. IEEE, 2011.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, page 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558602224.

M. Kull, M. Perello Nieto, M. Kängsepp, T. Silva Filho, H. Song, and P. Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

A. Kumar, S. Sarawagi, and U. Jain. Trainable calibration measures for neural networks from kernel mean embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2805–2814. PMLR, 10–15 Jul 2018.

A. Kumar, P. S. Liang, and T. Ma. Verified uncertainty calibration. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Le and X. Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

D. Lowell, Z. C. Lipton, and B. C. Wallace. Practical obstacles to deploying active learning. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 21–30, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1003.

A. Mazumdar and B. Saha. Clustering with noisy queries. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

T. Mitchell. Twenty Newsgroups. UCI Machine Learning Repository, 1999.

J. Moon, J. Kim, Y. Shin, and S. Hwang. Confidence-aware learning for deep neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 7034–7044, 2020.

J. Mukhoti, V. Kulharia, A. Sanyal, S. Golodetz, P. Torr, and P. Dokania. Calibrating deep neural networks using focal loss. *Advances in Neural Information Processing Systems*, 33:15288–15299, 2020.

R. Müller, S. Kornblith, and G. E. Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.

A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 625–632, 2005. ISBN 1595931805.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

H. Qiu, K. Chintalapudi, and R. Govindan. MCAL: Minimum cost human-machine active labeling. In *The Eleventh International Conference on Learning Representations*, 2023.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

A. J. Ratner, C. M. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain, 2016.

V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(43):1297–1322, 2010.

SGT. Aws sagemaker ground truth. https://aws.amazon.com/sagemaker/data-labeling/, 2022. Accessed: 2022-11-18.

C. Shin, W. Li, H. Vishwakarma, N. C. Roberts, and F. Sala. Universalizing weak supervision. In *International Conference on Learning Representations*, 2022.

A. Sorokin and D. Forsyth. Utility data annotation with amazon mechanical turk. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2008. doi: 10.1109/CVPRW.2008.4562953.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

H. Touvron, L. Martin, K. R. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. M. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. S. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. M. Kloumann, A. V. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023.

R. K. Vinayak and B. Hassibi. Crowdsourced clustering: Querying edges vs triangles. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 2016.

R. K. Vinayak, S. Oymak, and B. Hassibi. Graph clustering with missing data: Convex algorithms and analysis. *Advances in Neural Information Processing Systems*, 27, 2014.

R. K. Vinayak, T. Zrnic, and B. Hassibi. Tensor-based crowdsourced clustering via triangle queries. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2322–2326. IEEE, 2017.

H. Vishwakarma and F. Sala. Lifting weak supervision to structured prediction. In *Advances in Neural Information Processing Systems*, 2022.

H. Vishwakarma, H. Lin, F. Sala, and R. K. Vinayak. Promises and pitfalls of threshold-based auto-labeling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–213, 2001.

B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699, 2002.

F. Zhu, Z. Cheng, X.-Y. Zhang, and C.-L. Liu. Rethinking confidence calibration for failure prediction. In *European Conference on Computer Vision*, pages 518–536. Springer, 2022.

A Supplementary Material Organization

The supplementary material is organized as follows. We provide deferred details of the method in section B. Then, in section C, we provide additional experimental results and details of the experiment protocol and hyperparameters used for the experiments. Our code with instructions to run, is uploaded along with the paper.

A.1 Glossary

The notation is summarized in Table 3 below.

Symbol	Definition
$\mathbb{1}(E)$	indicator function of event E. It is 1 if E happens and 0 otherwise.
$\mathcal{X}$	feature space.
$\mathcal{Y}$	label space i.e. $1, 2, \dots k$.
$\mathcal{H}$	hypothesis space (model class for the classifiers).
$\mathcal{G}$	class of confidence functions.
k	number of classes.
$\mathbf{x}, y$	$\mathbf{x}$ is an element in $\mathcal{X}$ and y is its true label.
h	a hypothesis (model) in $\mathcal{H}$.
g	confidence function $g : \mathcal{X} \to \Delta^k$.
X_u	given pool of unlabeled data points.
$X_u^{(i)}$	unlabeled data left at the beginning of ith round.
$\hat{h}^{(i)}$	ERM solution and auto-labeling thresholds respectively in ith round.
$D^{(i)}_{\text{query}}$	labeled data queried from oracle (human) in the ith round.
$D^{(i)}_{\text{train}}$	training data to learn $\hat{h}^{(i)}$ in the ith round.
$D^{(i)}_{\text{val}}$	validation data in the ith round.
$D^{(i)}_{\text{cal}}$	calibration data in the ith round to learn a post-hoc g.
$D^{(i)}_{\text{th}}$	part of validation data in the ith round to estimate threshold $\mathbf{t}$.
$D^{(i)}_{\text{auto}}$	part of $X_u^{(i)}$ that got auto-labeled in the ith round.
D_{out}	Output labeled data, including auto-labeled and human labeled data.
$\mathbf{t}$	k dimensional vector of thresholds.
$\mathbf{t}[y]$	yth entry of $\mathbf{t}$ i.e. the threshold for class y.
$g(\mathbf{x})[y]$	the confidence score for class y output by confidence function g on data point $\mathbf{x}$.
$\hat{y}$	predicted class for data point $\mathbf{x}$.
f^*	unknown groundtruth labeling function.
N_u	number of unlabeled points, i.e. size of X_u.
N_t	number of manually labeled points that can be used for training h.
N_a	Total auto-labeled points in D_{out}.
ν	fraction of D_{val} that can be used for training post-hoc calibrator.
A	indices of points that are auto-labeled.
$X_u(A)$	subset of points in X_u with indices in A, i.e. the set of auto-labeled points.
$\tilde{y}_i$	label assigned to the ith point by the algorithm. It could be either y_i or $\hat{y}_i$.
y_i	groundtruth label for the ith point.
$\hat{y}_i$	predicted label for the ith point by classifier.
ϵ_a	auto-labeling error tolerance.
$\mathcal{E}(g, \mathbf{t} \mid h)$	population level auto-labeling error, see eq. (4).
$\mathcal{P}(g, \mathbf{t} \mid h)$	population level auto-labeling coverage, see eq. (3).
$\widehat{\mathcal{E}}(g, \mathbf{t} \mid h, D)$	estimated auto-labeling error, see eq. (6).
$\widehat{\mathcal{P}}(g, \mathbf{t} \mid h, D)$	estimated auto-labeling coverage, see eq. (5).
$\widetilde{\mathcal{E}}(g, \mathbf{t} \mid h, D)$	surrogate estimated auto-labeling error, see eq. (8).
$\widetilde{\mathcal{P}}(g, \mathbf{t} \mid h, D)$	surrogate estimated auto-labeling coverage, see eq. (7).

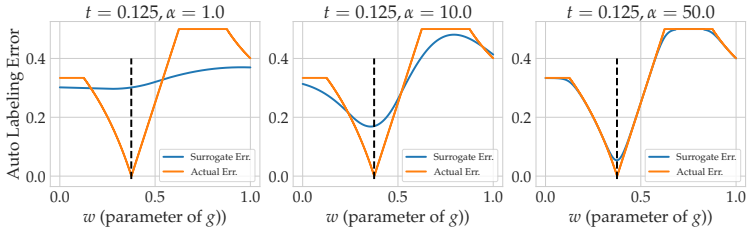
Table 3: Glossary of variables and symbols used in this paper.

B Appendix to Section 3

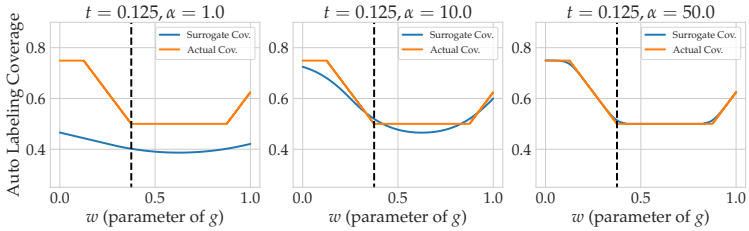
Tightness of surrogates. The surrogate auto-labeling error and coverage introduced to relax the optimization problem (P2) is indeed a good approximation of the actual auto-labeling error and coverage. To see this, we use a toy data setting of $x \sim \text{Uniform}(0, 1)$ with $1-$dimensional threshold classifier $h_\theta(x) = \mathbb{1}(x \geq \theta)$. For any x, let true labels $y = h_{0.5}(x)$ and consider the confidence function $g_w(x) = |w - x|$. Let $\hat{y} = h_{0.25}(x)$ and consider the points on the side where $\hat{y} = 1$. We plot actual and surrogate errors in Figure 5(a) and the surrogate and actual coverage in Figure 5(a).

for three choices of α. As expected, the gap between the surrogates and the actual functions diminishes as we increase the α.

Active Querying Strategy. We employ the margin-random query approach to select the next batch of training data. This method involves sorting points based on their margin (uncertainty) scores and selecting the top Cn_b points, from which n_b points are randomly chosen. This strategy provides a straightforward and computationally efficient way to balance the exploration-exploitation trade-off. It's important to acknowledge the existence of alternative active-querying strategies; however, we adopt the margin-random approach as our standard to maintain a focus on evaluating various choices of confidence functions for auto-labeling. Note that while we use the new confidence scores computed using post-hoc methods for auto-labeling, we do not use these scores in active querying. Instead, we use the softmax scores from the model for this. We do this to avoid conflating the study with the study of active querying strategies. We use $C = 2$ for all experiments.



(a) Auto-labeling error and surrogate error at various α.



(b) Auto-labeling coverage and surrogate coverage at various α.

Figure 5: Illustration of the tightness of surrogate error and coverage functions based on the choice of α.

C Additional Experiments and Details

C.1 Details of the experiment in section 2.2

We run TBAL for a single round on the CIFAR-10 dataset with a SimpleCNN classification model with around 5.8M parameters (Hussain, 2021). We randomly sampled 4,000 points for training the classifier and randomly sampled 1,000 points as validation data. We train the model to zero training error using minibatch SGD with learning rate 1e-3, weight decay 1e-3 Hanson and Pratt (1988); Krogh and Hertz (1991), momentum 0.9, and batch size 32. The trained model has validation accuracy around 55%, implying we could hope to get coverage around 55%. We run the auto-labeling procedure with an error tolerance of 5%.

C.2 Experiments on N_t, N_v and ν

We need to understand the effect of training data query budget i.e. N_t, the total validation data N_v, and the data that can be used for calibrating the model i.e. the calibration data fraction ν on the auto-labeling objective. As varying these hyperparameters on each train-time method is expensive, we experimented with only Squentropy as it was the best-performing method across settings for various datasets.

When we vary the budget for training data N_t, we observe from Figure 6 that our method does not require a lot of data to train the base model, i.e. achieving low auto-labeling error and high coverage with a low budget. While other methods benefit from having more training data for auto-labeling objectives, it comes at the expense of reducing the available data for validation.

From figure 7, we observe that, while the coverage of our method remains the same across different N_v, it reduces for other methods. The cause of this phenomenon can be attributed to the fact that we are borrowing the data from the training budget as it limits the performance of the base model, which in turn limits the auto-labeling objective.

As we increase the percentage of data that can be used to calibrate the model, i.e., ν, we note from figure 8 that other methods improve the coverage, which can be understood from the fact that when more data is available for calibrating the model, the model becomes better in terms of the auto-labeling objective. But it's interesting to note that even with a low calibration fraction, our method achieves superior coverage compared to other methods. It is also important to note that the auto-labeling error increases as we increase ν. This is because when ν increases, the number of data points used to estimate the threshold decreases, leading to a less granular and precise threshold.

Feature	Model	Error	Coverage
Pre-logits	Two Layer	4.6 ± 0.3	82.8 ± 0.5
Logits	Two Layer	3.2 ± 1.3	82.8 ± 0.3
Concat	Two Layer	3.3 ± 0.8	82.9 ± 0.4

Table 4: Auto-labeling error and coverage for the 3 feature representations we could use for 20 Newsgroup. As we can see, the feature representation does not lead to a significant difference in auto-labeling error and coverage.

Feature	Model	Error	Coverage
Pre-logits	Two Layer	2.1 ± 0.5	79.0 ± 0.2
Logits	Two Layer	3.1 ± 0.4	76.5 ± 0.9
Concat	Two Layer	2.3 ± 0.5	79.0 ± 0.3

Table 5: Auto-labeling error and coverage for the 3 feature representations we could use for CIFAR10 SimpleCNN. As we can see, the feature representation does not lead to a significant difference in auto-labeling error and coverage.

C.3 Experiments on `Colander` input

Figure 4 illustrates that we could use logits (last layer's representations), pre-logits (second last layer's representations), or the concatenation of these two as the input to g. To help us decide which one we should use, we conduct a hyperparameter search for input features on the CIFAR-10 and 20 Newsgroup dataset using the Squentropy train-time method. Table 4 and 5 present the auto-labeling error and coverage of using the 3 types of feature representations. As we can see, all feature representation leads to a similar auto-labeling error and coverage, and in some cases, it is better to include pre-logits as well. Therefore, we use concatenated representation (Concat), allowing more flexibility.

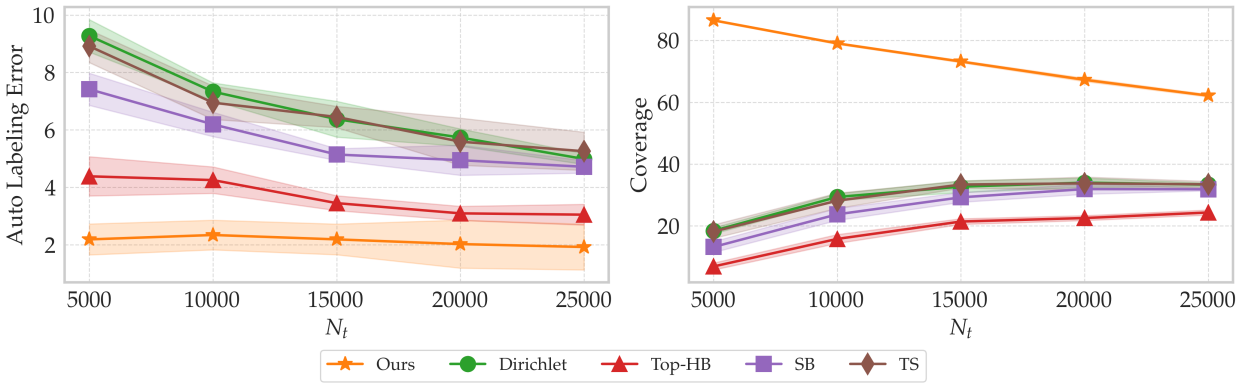


Figure 6: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various N_t

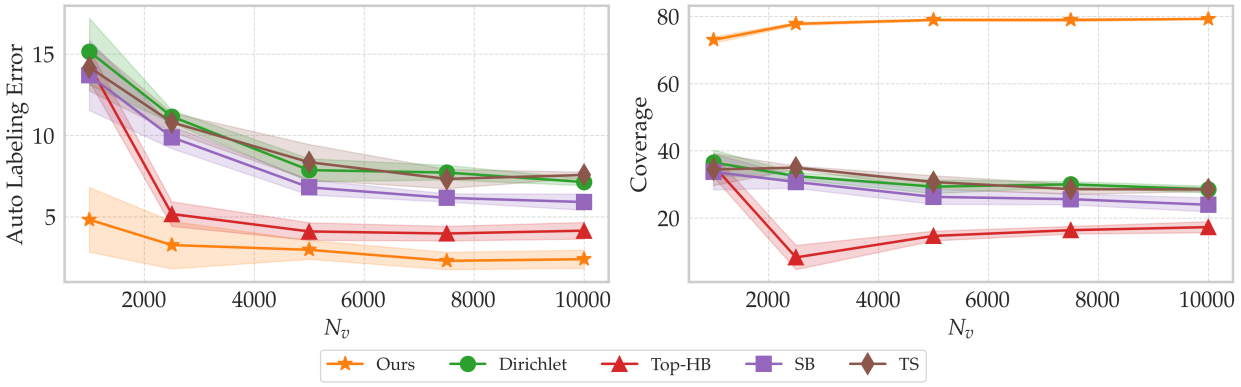


Figure 7: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various N_v

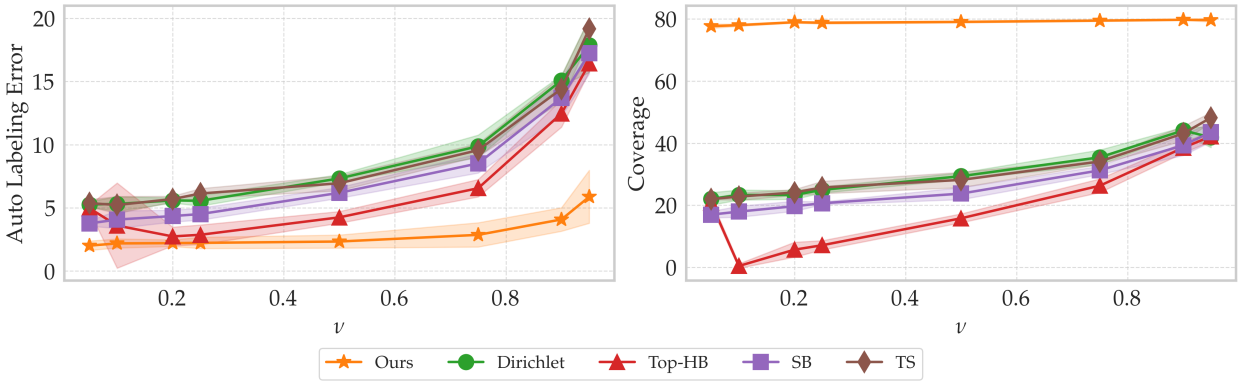


Figure 8: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various ν

C.4 Hyperparameters

The hyperparameters and their values we swept over are listed in Table 6 and 7 for train-time and post-hoc methods, respectively.

C.5 Train-time and post-hoc methods

C.5.1 Train-time methods

1. *Vanilla*: Neural networks are commonly trained by minimizing the cross entropy loss using stochastic gradient descent (SGD) with momentum (Amari, 1993; Bottou, 2012). We refer to this as the Vanilla training method. We also include weight decay to mitigate the overconfidence issue associated with this method (Guo et al., 2017).

Method	Hyperparameter	Values
Common	optimizer	SGD
	learning rate	0.001, 0.01, 0.1
	batch size	32, 256
	max epoch	50, 100
	weight decay	0.001, 0.01, 0.1
	momentum	0.9
CRL	rank target	softmax
	rank weight	0.7, 0.8, 0.9
FMFP	optimizer	SAM

Table 6: Hyperparameters swept over for train-time methods. Those listed next to Common are the hyperparameters for the four train-time methods: Vanilla, CRL, FMFP, and Squentropy. Therefore, we do not list those again for each method. Note that for FMFP, we used SAM optimizer instead of SGD. For each method, we swept through all possible combinations of the possible values for each hyperparameter. Underlined values are only used on TinyImageNet since it is a complicated dataset containing 200 classes.

2. *Squentropy* (Hui et al., 2023): This method adds the average square loss over the incorrect classes to the cross-entropy loss. This simple modification to the Vanilla method leads to the end model with better test accuracy and calibration.
3. *Correctness Ranking Loss (CRL)* (Moon et al., 2020): This method includes a term in the loss function of the vanilla training method so that the confidence scores of the model are aligned with the ordinal rankings criterion (Hendrycks and Gimpel, 2017; Corbière et al., 2019). The confidence functions satisfying this criterion produce high scores on points where the probability of correctness is high and low scores on points with low probabilities of being correct.
4. *FMFP* (Zhu et al., 2022) aims to align confidence scores with the ordinal rankings criterion. It uses Sharpness Aware Minimizer (SAM) (Foret et al., 2021) to train the model, with the expectation that the flat minima would benefit the ordinal rankings objective of the confidence function.

C.5.2 Post-hoc methods

1. *Temperature scaling* (Guo et al., 2017): This is a variant of Platt scaling (Guo et al., 2017), a classic and one of the easiest parametric methods for post-hoc calibration. It rescales the logits by a learnable scalar parameter and has been shown to work well for neural networks.
2. *Top-Label Histogram-Binning* (Gupta and Ramdas, 2022): Since TBAL assigns the top labels (predicted labels) to the selected unlabeled points, it is appealing to only calibrate the scores of the predicted label. Building upon a rich line of histogram-binning methods (non-parametric) for post-hoc calibration (Zadrozny and Elkan, 2002), this method focuses on calibrating the scores of predicted labels.
3. *Scaling-Binning* (Kumar et al., 2019): This method combines parametric and non-parametric methods. It first applies temperature scaling and then bins the confidence function values to ensure calibration.
4. *Dirichlet Calibration* (Kull et al., 2019): This method models the distribution of predicted probability vectors separately on instances of each class and assumes the class conditional distributions are Dirichlet distributions with different parameters. It uses linear parameterization for the distributions, which allows easy implementation in neural networks as additional layers and softmax output.

Note: For binning methods, uniform mass binning (Zadrozny and Elkan, 2002) has been a better choice over uniform width binning. Hence, we use uniform mass binning as well.

C.6 Compute resources

Our experiments were conducted on machines equipped with the NVIDIA RTX A6000 and NVIDIA GeForce RTX 4090 GPUs.

C.7 Detailed dataset and model

1. The MNIST dataset LeCun (1998) consists of 28×28 grayscale images of hand-written digits across 10 classes. It was used alongside the LeNet5 LeCun et al. (1998), a convolutional neural network, for auto-labeling.

2. The CIFAR-10 dataset Krizhevsky et al. (2009) contains $3 \times 32 \times 32$ color images across 10 classes. We utilized its raw pixel matrix in conjunction with SimpleCNN Hussain (2021), a convolutional neural network with approximately 5.8M parameters, for auto-labeling.
3. Tiny-ImageNet Le and Yang (2015) is a color image dataset that consists of 100K images across 200 classes. Instead of using the $3 \times 64 \times 64$ raw pixel matrices as input, we utilized CLIP Radford et al. (2021) to derive embeddings within the $\mathbb{R}^{512}$ vector space. We used a 3-layer perceptron (1,000-500-300) as the auto-labeling model.
4. 20 Newsgroups Mitchell (1999); Pedregosa et al. (2011) is a natural language dataset comprising around 18,000 news posts across 20 topics. We used the FlagEmbedding Xiao et al. (2023) to map the textual data into $\mathbb{R}^{1024}$ embeddings. We used a 3-layer perceptron (1,000-500-30) as the auto-labeling model.

C.8 Detailed experiments protocol

We predefined TBAL hyperparameters for each dataset-model pair and the hyperparameters we will sweep for each train-time and post-hoc method in Table 6 and Table 7 respectively. For a dataset-model pair, initially, we perform a hyperparameter search for the train-time method. Subsequently, we optimize the hyperparameters for post-hoc methods while keeping the train-time method fixed with the previously found optimum hyperparameter for that dataset-model pair.

We fix the hyperparameters for the train-time method while searching hyperparameters for the post-hoc method to alleviate computational budget throttle. We effectively reduce the search space to the sum of the cardinalities of unique hyper-parameter combinations across the two methods instead of a larger multiplicative product. Furthermore, due to the independent nature of these hyper-parameter combinations, TBAL runs can be highly parallelized to expedite the search process.

Since TBAL operates iteratively to acquire human labels for model training, selecting hyper-parameters at each round of TBAL could quickly become intractable and lose its practical significance. To better align with its practical usage, we only conducted a hyperparameter search for the initial TBAL round. The specific set of hyperparameters used for the search are reported in Table 7.

After completing the hyperparameter search for train-time and post-hoc methods, the determined hyperparameter combinations are subjected to a full evaluation across all iterations of TBAL. At the end of each iteration, the auto-labeled points are evaluated against their ground truth labels to determine their auto-labeling error. These points are then added to the auto-labeled set, where their ratio to the total amount of unlabeled data determines the coverage. This iterative process continues until all unlabeled data are exhaustively labeled by either the oracle or through auto-labeling in the final iteration. The auto-labeling error and coverage at the final iteration of TBAL are then recorded.

Since TBAL incorporates randomized components as detailed in Algorithm 1, we ran the algorithm 5 times, each with a unique random seed while maintaining the same hyperparameter combination. We then recorded the results from the final iteration of these runs and calculated the mean and standard deviation of both auto-labeling error and coverage. These figures are reported in Table 2.

A limitation of the grid search approach in hyper-parameter optimization becomes apparent when our predefined hyper-parameter choices result in sub-optimal coverage and auto-labeling errors. Using these sub-optimal hyper-parameters can adversely affect the multi-round iterative process in TBAL, prompting the need for repetitive searches to find more effective hyper-parameters. When encountering such scenarios, TBAL users should explore additional hyper-parameter options until satisfactory performance is achieved in the initial round. However, we opted for a more straightforward approach to hyper-parameter selection, mindful of the computational demands of repeatedly optimizing multiple hyper-parameters across different methods. In scenarios expressed conditionally, we retained the top-1 hyper-parameter combination for any given method if it achieved the highest coverage while adhering to the specified error margin (ϵ_a). If no hyper-parameter combinations yielded an auto-labeling error at most equal to the error margin (ϵ_a), we then chose the hyper-parameter combination with the lowest auto-labeling error, regardless of its coverage. In the case of ties, we resolved them through random selection. This process results in obtaining singular values for each choice of hyper-parameter after completing each method's hyper-parameter search.

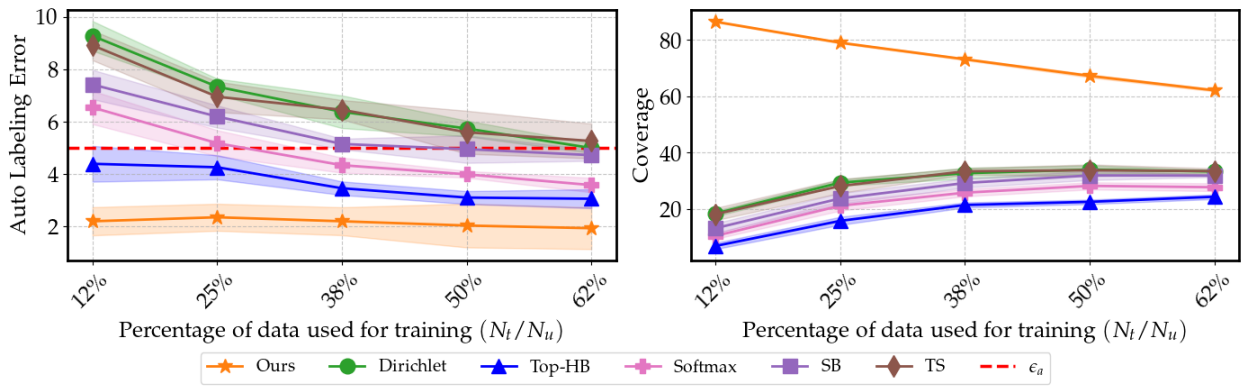


Figure 9: Auto-labeling error and coverage for different post-hoc methods on CIFAR-10 while we vary N_t. $N_u = 40,000$ is the size of the given unlabeled pool.

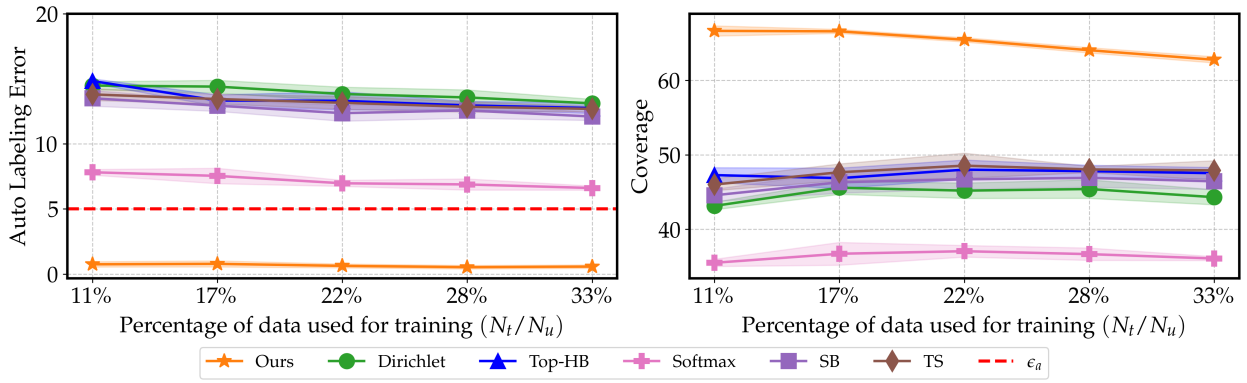


Figure 10: Auto-labeling error and coverage for different post-hoc methods on Tiny-ImageNet while we vary N_t. $N_u = 90,000$ is the size of the given unlabeled pool.

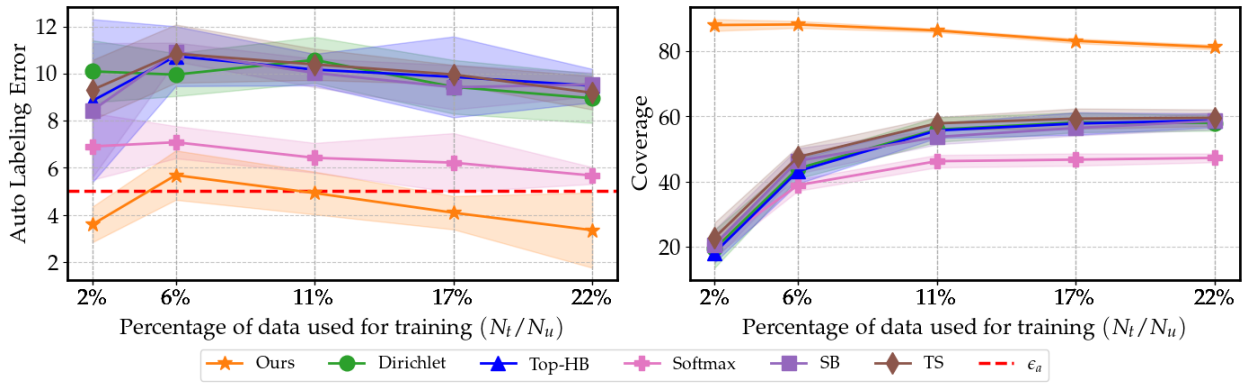


Figure 11: Auto-labeling error and coverage for different post-hoc methods on 20 Newsgroups while we vary N_t. $N_u = 9,052$ is the size of the given unlabeled pool.

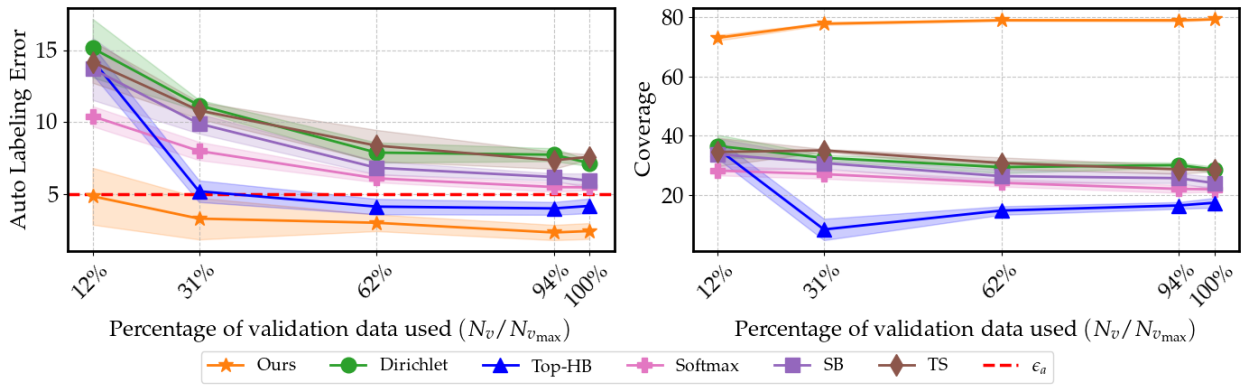


Figure 12: Auto-labeling error and coverage for different post-hoc methods on CIFAR-10 while we vary N_v. $N_{v_{\max}} = 8,000$ is the maximum number of points available for validation.

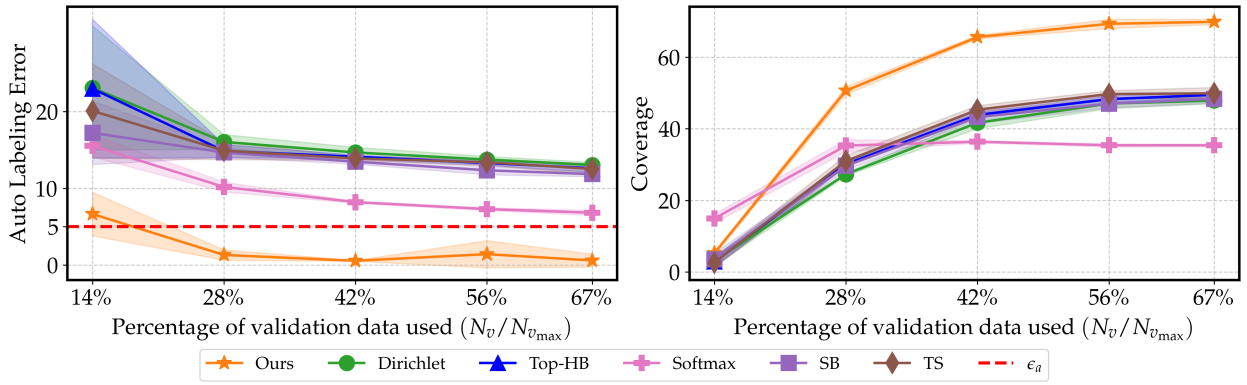


Figure 13: Auto-labeling error and coverage for different post-hoc methods on Tiny-ImageNet while we vary N_v. $N_{v_{\max}} = 18,000$ is the maximum number of points available for validation.

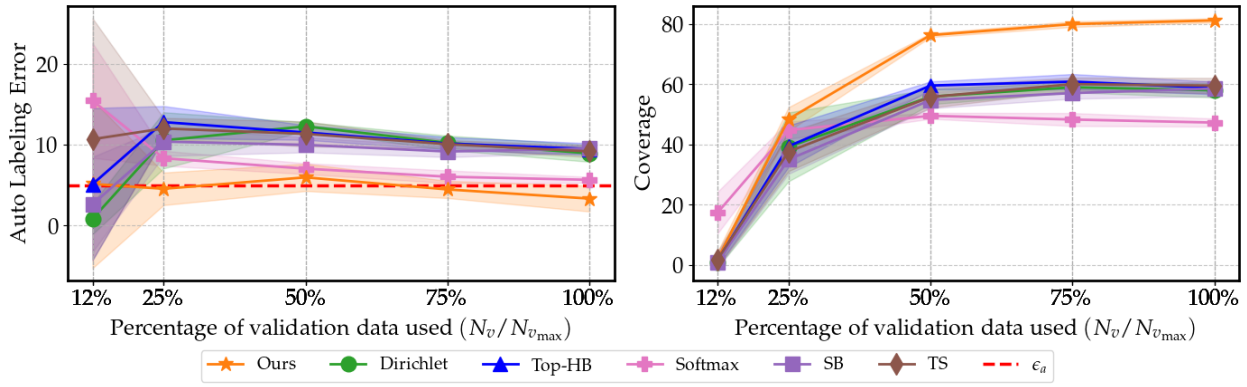


Figure 14: Auto-labeling error and coverage for different post-hoc methods on 20 Newsgroups while we vary N_v. $N_{v_{\max}} = 1,600$ is the maximum number of points available for validation.

Method	**Hyperparameter**	**Values**
Temperature scaling	optimizer	Adam
	learning rate	0.001, 0.01, 0.1
	batch size	64
	max epoch	500
	weight decay	0.01, 0.1, 1
Top-label histogram binning	points per bin	25, 50
Scaling-binning	number of bins	15, 25
	learning rate	0.001, 0.01, 0.1
	batch size	64
	max epoch	500
	weight decay	0.01, 0.1, 1
Dirichlet calibration	regularization parameter	0.001, 0.01, 0.1
Ours	λ	10, 100
	features key	concat
	class-wise	independent
	optimizer	Adam
	learning rate	0.01, 0.1
	max epoch	500
	weight decay	0.01, 0.1, 1
	batch size	64
	regularize	false
	α	0.01, 0.1, 1

Table 7: Hyperparamters swept over for post-hoc methods. For each method, we swept through all possible combinations of the possible values for each hyperparameter.