Pearls from Pebbles: Improved Confidence Functions for Auto-labeling

Anonymous Author(s)
Affiliation
Address
email

Abstract

Auto-labeling techniques produce labeled data with minimal manual annotations using the representations from self-supervised models and confidence scores. A popular technique, threshold-based auto-labeling (TBAL) trains model using these representations and manual annotations, and assigns model's prediction as label to the points where model's confidence score is greater than certain threshold. However, the model's scores can be overconfident and lead to poor performance. We show that, calibration, a common remedy for the overconfidence problem, falls short in tackling this problem for TBAL. Thus, instead of using existing calibration methods, we introduce a framework for optimal confidence functions for TBAL and develop `Colander`, a method designed to maximize auto-labeling performance. We perform an extensive empirical evaluation of `Colander` and other confidence functions, using representations from CLIP and text embedding models for image and text data respectively. We find `Colander` achieves up to 60% improvement on coverage (the proportion of points labeled by model) over the baselines while maintaining error level below 5% and using the same amount of labeled data.

1 Introduction

The demand for labeled data in machine learning (ML) is perpetual. Threshold-based auto-labeling (TBAL) is a promising solution to obtain high-quality labeled data at low cost [41, 37, 49] using model's confidence scores and self-supervision. It powers industry products like Amazon SageMaker Ground Truth [41]. The confidence function is critical to the TBAL workflow. Existing TBAL systems rely on common choices like softmax outputs from neural networks [37, 49]. These functions *are not well aligned with the objective of the auto-labeling system*. Using them results in substantially suboptimal coverage (Figure 1(a)). For this reason, we ask:

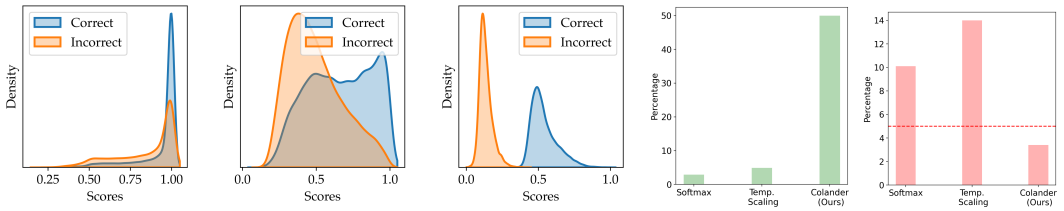
> What are the right choices of confidence functions for TBAL and how can we obtain them?

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.

Overconfidence further stymies hopes of balancing accuracy and coverage. While overconfidence is a challenge in general, it is exacerbated in TBAL: since models are trained on a small amount of labeled data, making the problem of designing confidence functions even more challenging. Figure 1(a) shows that softmax scores are overconfident, resulting in poor auto-labeling performance.

Several methods have been introduced to address overconfidence [10]. Applying these miss out on significant performance (Figure 1(b)), since the calibration goal differs from auto-labeling. From the auto-labeling standpoint, we seek minimum overlap between the correct and incorrect model prediction scores.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.



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

Figure 1: Scores distributions (Kernel Density Estimates) of a CNN model trained on CIFAR-10 data. (a) softmax scores of the 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 1000 validation points (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.

We tackle these challenges by ***proposing a framework to learn suitable confidence functions*** for TBAL. 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-world datasets, 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

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 (feature) space $\mathcal{X}$. This could either be the space of raw features or the representation space from some self-supervised model. Let $\mathcal{Y} = \{1, \ldots, k\}$ be the label space. There is an unknown ground truth 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 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. Table 2 contains a summary of the notation.

Threshold-based Auto-labeling. Threshold-based auto-labeling (TBAL) 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 $\mathbf{x} \in X_u$, a label $\tilde{y} \in \mathcal{Y}$. The output label could 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 $\widehat{\mathcal{P}}(X_u(A))$, are defined as follows:

$$\widehat{\mathcal{E}}(X_u(A)) := \frac{1}{N_a} \sum_{i \in A} \mathbb{1}(\tilde{y}_i \neq f^*(\mathbf{x}_i)), \quad \text{and} \quad \widehat{\mathcal{P}}(X_u(A)) := |A|/N_u = N_a/N_u. \tag{1}$$

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 $\epsilon_a \in [0, 1]$. 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.

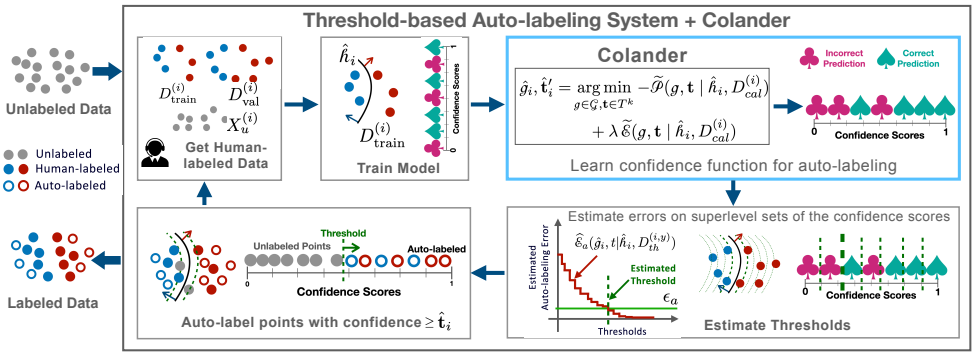


Figure 2: Threshold-based Auto-labeling with `Colander`: takes unlabeled data as input, selects a small subset of data points, and obtains human labels for them to create $D^{(i)}_{\text{train}}$ and $D^{(i)}_{\text{val}}$ for the ith iteration. Trains model $\hat{h}_i$ on $D^{(i)}_{\text{train}}$. In contrast to the standard TBAL procedure, here we randomly split $D^{(i)}_{\text{val}}$ into two parts, $D^{(i)}_{\text{cal}}$ and $D^{(i)}_{\text{th}}$. `Colander` kicks in, takes $\hat{h}_i$ and $D^{(i)}_{\text{cal}}$ as input and learns a coverage maximizing confidence function $\hat{g}_i$ for $\hat{h}_i$. Using $D^{(i)}_{\text{th}}$ and $\hat{g}_i$ auto-labeling thresholds $\hat{\mathbf{t}}_i$ are determined to ensure the auto-labeled data has error at most ϵ_a. After obtaining the thresholds the rest of the steps are the same as standard TBAL. The whole workflow runs until all the data is labeled or another stopping criterion is met.

3 Proposed Method (`Colander`)

3.1 Auto-labeling optimization framework

In any iteration of TBAL, we have a model $\hbar$ 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, we propose a framework to find the right function from a sufficiently rich family.

Optimization problem. 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:

$$\widetilde{\mathscr{P}}(g, \mathbf{t} | \hbar, D_{\text{cal}}) := \frac{1}{|D_{\text{cal}}|} \sum_{(\mathbf{x},y)\in D_{\text{cal}}} \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big), \tag{2}$$

$$\widetilde{\mathscr{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) := \frac{\sum_{(\mathbf{x},y)\in D_{\text{cal}}} \mathbb{1}\big(y \neq \hat{y}\big)\, \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big)}{\sum_{(\mathbf{x},y)\in D_{\text{cal}}} \sigma\big(\alpha, g(\mathbf{x})[\hat{y}] - \mathbf{t}[\hat{y}]\big)}, \tag{3}$$

and the surrogate optimization problem as follows,

$$\underset{g\in\mathcal{G}, \mathbf{t}\in T^k}{\arg\min} \quad -\widetilde{\mathscr{P}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) + \lambda\, \widetilde{\mathscr{E}}(g, \mathbf{t} \mid \hbar, D_{\text{cal}}) \tag{P1}$$

Here, $\lambda \in \mathbb{R}^+$ is the penalty term controlling the relative importance of the auto-labeling error and coverage. We tune it with the procedure discussed in Appendix B.4. The gap between the surrogate and actual coverage diminishes as $\alpha \to \infty$. We discuss this in the Appendix. Our framework is flexible with respect to the choice of $\mathcal{G}$. We use 2-layer neural nets for $\mathcal{G}$ and use representations from the last two layers of $\hat{h}$ as input for g. See Appendix for further details.

Solving the surrogate optimization. The optimization problem (P1) is non-convex. Nevertheless, it is differentiable and we can apply gradient-based methods. We solve for g and $\mathbf{t}$ simultaneously using Adam [19]. Appendix B.4 lists the detailed procedure of our method in a TBAL system.

4 Empirical Evaluation

We conduct extensive empirical evaluation with variety of train-time and post-hoc calibration methods, and feature choices. We use *Vanilla* training with SGD [1, 3, 10]. *Squentropy* [16], *Correctness*

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 1: 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.

Ranking Loss (CRL) [30] and *FMFP* [53] as train-time methods. We pipe these with `Colander` and other post-hoc methods *Temperature scaling* [10], *Top-Label Histogram-Binning* [12], *Scaling-Binning* [24] and *Dirichlet Calibration* [22]. We use raw features for *MNIST* [26] and *CIFAR-10* [20], CLIP [38] embeddings for *Tiny-ImageNet* and FlagEmbedding [50] for *20 Newsgroups* [29] The results are reported in Table 1.

TBAL with `Colander` consistently achieves significantly higher coverage compared to vanilla training and softmax scores, especially on more complex datasets like Tiny-ImageNet, outperforming baseline models across all data settings. Post-hoc calibration techniques slightly improve coverage but at the cost of higher error, as their goal of mitigating overconfidence is not aligned with TBAL's needs. Importantly, `Colander` is compatible with different train-time methods and amplifies performance gains, particularly when paired with Squentropy, where coverage increases by 6-7% and error decreases, outperforming other train-time approaches. In contrast, methods focused on ordinal ranking objectives, such as CRL and FMFP, perform poorly in the TBAL setting due to challenges like limited training data and reduced effectiveness in differentiating between correct and incorrect predictions when training error reaches zero after a few rounds.

5 Conclusion

We studied confidence scoring functions used in threshold-based auto-labeling (TBAL) and showed that the commonly used choices 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 `Colander` extensively against common baselines on several real-world datasets and found that it improves the performance of TBAL significantly. A limitation of `Colander` is that, similar to other post-hoc methods it also requires validation data to learn the scores. Reducing (or eliminating) this dependence could be an interesting future work.

References

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

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

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

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

[27] 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.

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

[29] T. Mitchell. Twenty Newsgroups. UCI Machine Learning Repository, 1999.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

[41] SGT. Aws sagemaker ground truth. https://aws.amazon.com/sagemaker/data-labeling/, 2022. Accessed: 2024-05-22.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

Supplementary Material Organization

The supplementary material is organized as follows. We provide deferred details of background and motivation section in Appendix A of the method in Appendix B. Then, in Appendix 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.

Contents

A Appendix to Section 2

A.1 Detailed Comparison with Active Learning and Self Training

To illustrate the differences between TBAL and the combination of Active Learning (AL) and Self-Training for the task of data labeling, we run an experiment on the 2 concentric circles data setting as used in [49]. The details are as follows:

Data setting. We generate two concentric circles with points in the outer circle belonging to one class and the inner circle belonging to the other class. The total number of points generated is 10,000 of which we use 2000 for validation.

Methods. We run TBAL, AL+Self-Training, and AL+Self-Training+SC, using logistic regression. The combination of AL+Self-Training means, in each iteration, the algorithm queries human-labeled data points and pseudo-labels the points in the unlabeled data using self-training and adds both the human-labeled and pseudo-labeled points in the training pool. With this procedure, AL+Self-Training first learns the best classifier ($\hat{h}_{\mathrm{al-st}}$) with the given budget of maximum training points (N_t) that can be queried from humans. Then it auto-labels all the remaining unlabeled points with this classifier's predictions. For AL+Self-Training+SC, we do selective auto-labeling using $\hat{h}_{\mathrm{al-st}}$, i.e., only auto-label the points where the classifier will have an error at most ϵ_a. We use ϵ_a =1% here.

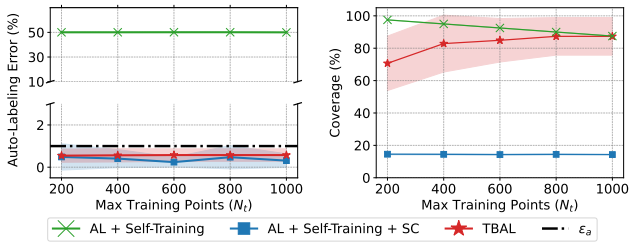


Figure 3: Results of experiment on 2-concentric circles to show the differences between TBAL, AL and ST.

Results and Discussion. The Figure 3 shows auto-labeling error and coverage achieved by these methods when run with different choices of human-labeled data budget for training. First, we can see that even with linear classifiers TBAL is able to auto-label a huge chunk of the data (high coverage) while maintaining auto-labeling error below the tolerance level of 1% On the other hand, methods like AL+Self-Training (+SC) that try to first learn the optimal classifier in the given function class either have high auto-labeling error or very low coverage. These results are also consistent with the observations in [49] on the comparison between TBAL and AL, AL+SC. While such findings confirm the notion that there are differences—and, at least in some settings, advantages—for the TBAL approach compared to other techniques, we reiterate that our goal is to understand and improve the role of the confidence function within TBAL, rather than comparing TBAL to other techniques.

A.2 Details of the motivating experiment in Section 2

We run TBAL for a single round on the CIFAR-10 dataset with a SimpleCNN classification model with around 5.8M parameters [17]. 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 [13, 21], 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%.

B Appendix on Our Method

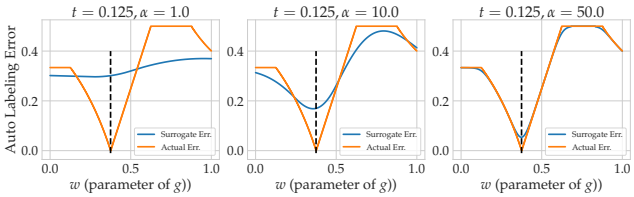
B.1 Detailed Algorithms

See Algorithms 1 and 2.

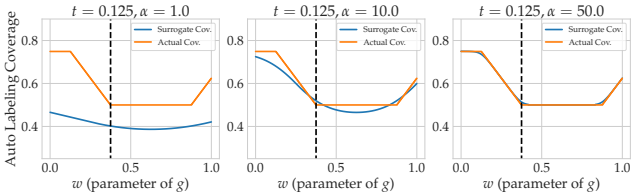
B.2 Tightness of surrogates.

The surrogate auto-labeling error and coverage introduced to relax the optimization problem (**??**) 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 4(a) and the surrogate and actual coverage in Figure 4(a).

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



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



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

Figure 4: Illustration of the tightness of surrogate error and coverage functions based on the choice of α.

B.3 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.

B.4 TBAL procedure with `Colander`

We take the workflow of TBAL and plugin our method `Colander` to learn the new confidence function and threshold. We discuss the updated workflow below and place the detailed Algorithms 1 and 2 in the Appendix B due to space constraints.

1. Initialization. First select n_s points randomly from X_u and obtain human labels for them to create initial training data $D^{(1)}_{\text{train}}$. This is written as $\text{RandomQuery}(X_u, n_s)$ in Algorithm 1. The procedure $\text{RandomQuery}(X_u, n_s)$ selects n_s points randomly from X_u and obtains human labels for them to create $D^{(1)}_{\text{train}}$.

2. Train classification model. After obtaining human-labeled training data $D^{(i)}_{\text{train}}$ for the current round i, the procedure $\text{TrainModel}(\mathcal{H}, D^{(i)}_{\text{train}})$ trains a model from model class $\mathcal{H}$ on the training data $D^{(i)}_{\text{train}}$. Any training procedure can be used here. We use methods listed in Section **??** for model training. This step outputs a model $\hat{h}_i$ trained on $D^{(i)}_{\text{train}}$.

3. Learn new confidence function using `Colander`. The model $\hat{h}_i$ obtained in the previous step also produces softmax scores that can be used to for auto-labeling. However, as we saw earlier in Section 2, using these scores may lead to poor auto-labeling performance. Thus, we plug in our procedure `Colander` to learn new scores that are designed to maximize the auto-labeling performance. We first randomly splits the validation data $D_{\text{val}}^{(i)}$ into $D_{\text{cal}}^{(i)}$ and $D_{\text{th}}^{(i)}$ using procedure RANDOMSPLIT$(D_{\text{val}}^{(i)}, \nu)$. The part $D_{\text{cal}}^{(i)}$ has a fraction ν of the points from $D_{\text{val}}^{(i)}$. Then we consider problem P1 with $\hat{h}_i$ and $D_{\text{cal}}^{(i)}$. We solve it to obtain the post-hoc confidence function $\hat{g}_i$.

4. Threshold estimation. The scores from the new confidence function $\hat{g}_i$ on $D_{\text{th}}^{(i)}$ are used to estimate auto-labeling thresholds in Algorithm 2. This procedure finds thresholds for each class separately. It first splits the points in $D_{\text{th}}^{(i)}$ according to the ground truth class into subsets $D_{\text{th}}^{(i,y)}$. Then, for each class y, it finds the auto-labeling threshold $\hat{\mathbf{t}}[y]$ by selecting the minimum threshold t such that the estimate of auto-labeling error plus a confidence interval, estimated on points in $D_{\text{th}}^{(i,y)}$ having scores above t, is at most the given error tolerance ϵ_a. 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.

5. Auto-labeling. This is a simple step. We computes the scores on the remaining unlabeled data $X_u^{(i)}$ using the function $\hat{g}_i$ and any point $\mathbf{x} \in X_u^{(i)}$ having score above $\hat{\mathbf{t}}[\hat{y}]$ is assigned auto-label $\hat{y} = \hat{h}_i(\mathbf{x})$, and the points that did not meet this criterion remain unlabeled.

6. Remove auto-labeled points. The points that got auto-labeled in the previous steps are removed from the unlabeled pool. To make the validation data consistent with this unlabeled pool for the next round, the points in the validation data that fall into the auto-labeling region are also removed.

7. Get more human-labeled data. Lastly, it calls the procedure ACTIVEQUERY$(\hat{h}_i, X_u^{(i)}, n_b)$ to select n_b points from the remaining unlabeled pool using an active learning strategy. This newly acquired human-labeled data is added into the training data $D_{\text{train}}^{(i)}$. The details of the querying strategy are in Appendix B. The procedure then moves to step 2 and runs the loop until there are no more unlabeled points left or it has queried the stipulated number of human-labels N_t.

B.5 Glossary

The notation is summarized in Table 2 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. $1, 2, \ldots k$.
$\mathcal{H}$	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.
$D_{\text{query}}^{(i)}$	labeled data queried from oracle (human) in the ith round.
$D_{\text{train}}^{(i)}$	training data to learn $\hat{h}^{(i)}$ in the ith round.
$D_{\text{val}}^{(i)}$	validation data in the ith round.
$D_{\text{cal}}^{(i)}$	calibration data in the ith round to learn a post-hoc g.
$D_{\text{th}}^{(i)}$	part of validation data in the ith round to estimate threshold $\mathbf{t}$.
$D_{\text{auto}}^{(i)}$	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. (**??**).
$\mathcal{P}(g, \mathbf{t} \mid h)$	population level auto-labeling coverage, see eq. (**??**).
$\widehat{\mathcal{E}}(g, \mathbf{t} \mid h, D)$	estimated auto-labeling error, see eq. (**??**).
$\widehat{\mathcal{P}}(g, \mathbf{t} \mid h, D)$	estimated auto-labeling coverage, see eq. (**??**).
$\widetilde{\mathcal{E}}(g, \mathbf{t} \mid h, D)$	surrogate estimated auto-labeling error, see eq. (3).
$\widetilde{\mathcal{P}}(g, \mathbf{t} \mid h, D)$	surrogate estimated auto-labeling coverage, see eq. (2).

Table 2: Glossary of variables and symbols used in this paper.

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. ***/
    $\widetilde{D}^{(i)}_u \leftarrow \{(\mathbf{x}, \hat{h}_i(\mathbf{x})) : \mathbf{x} \in X^{(i)}_u\}$
    $D^{(i)}_{\text{auto}} \leftarrow \{(\mathbf{x}, \hat{y}) \in \tilde{D}^{(i)}_u : \hat{g}_i(\mathbf{x})[\hat{y}] \geq \hat{\mathbf{t}}_i[\hat{y}]\}$
    $X^{(i)}_u \leftarrow X^{(i)}_u \setminus \{\mathbf{x} : (\mathbf{x}, \hat{y}) \in D^{(i)}_{\text{auto}}\}$ ▷Remove auto-labeled points from the unlabeled pool.
    $\widetilde{D}^{(i)}_{\text{val}} \leftarrow \{(\mathbf{x}, \hat{h}_i(\mathbf{x})) : (\mathbf{x}, y) \in D^{(i)}_{\text{val}}\}$
    $D^{(i+1)}_{\text{val}} \leftarrow \{(\mathbf{x}, \hat{y}) \in \tilde{D}^{(i)}_{\text{val}} : \hat{g}_i(\mathbf{x})[\hat{y}] < \hat{\mathbf{t}}_i[\hat{y}]\}$ ▷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

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. ***/
    $T'_y \leftarrow \{t \in T : \widehat{\mathcal{P}}\big(\hat{g}_i, t \mid \hat{h}_i, D_{\text{th}}^{(i,y)}\big) \geq \rho_0\} \cup \{\infty\}$
    ▷ /*** 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

C Additional Experiments and Details

Choice of $\mathcal{G}$. Our framework is flexible with respect to the choice of function class $\mathcal{G}$. In this work, we use neural networks with at least two layers on model class $\mathcal{H}$. We use representations from the last two layers of as input for the functions in $\mathcal{G}$. Let $\mathbf{z}^{(1)}(\mathbf{x}; h) \in \mathbb{R}^k$ and $\mathbf{z}^{(2)}(\mathbf{x}; h) \in \mathbb{R}^{d_2}$ be the outputs of the last and the second-last layer of the net h for input $\mathbf{x}$ and let $\mathbf{z}(\mathbf{x}; h) := [\mathbf{z}^{(1)}(\mathbf{x}; h), \mathbf{z}^{(2)}(\mathbf{x}; h)]$ denote the concatenation. This input is passed to network $\mathcal{G}_{nn_2} : \mathbb{R}^{k+d_2} \mapsto \Delta^k$; it outputs confidence scores for the k classes. Specifically g is defined as $g(\mathbf{x}) := \texttt{softmax}\big(\mathbf{W}_2 \texttt{tanh}(\mathbf{W}_1 \mathbf{z}(\mathbf{x}; h))\big)$. 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. As usual, for $\mathbf{v} \in \mathbb{R}^d$, $\texttt{softmax}(\mathbf{v})[i] := \exp(\mathbf{v}[i])/(\sum_j \exp(\mathbf{v}[j]))$ and $\texttt{tanh}(\mathbf{v})[i] := (\exp(2\mathbf{v}[i]) - 1)/(\exp(2\mathbf{v}[i]) + 1)$.

C.1 Experiments on N_t, N_v and ν

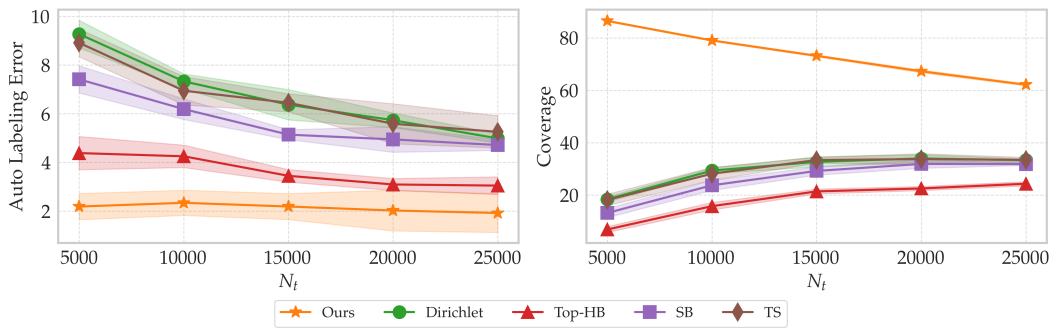


Figure 5: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various N_t

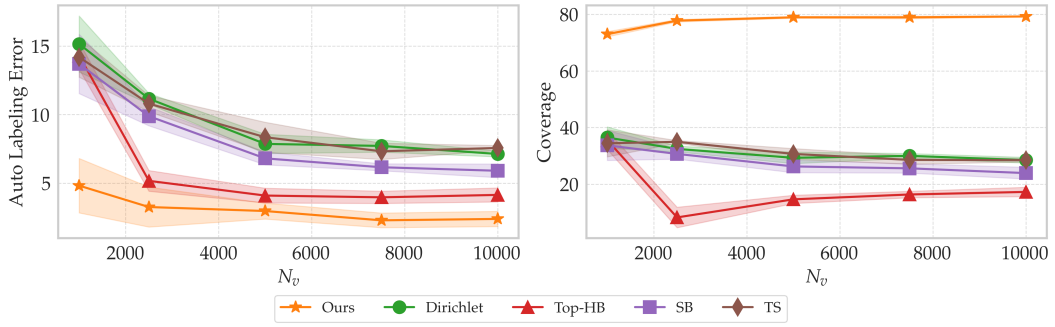


Figure 6: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various N_v

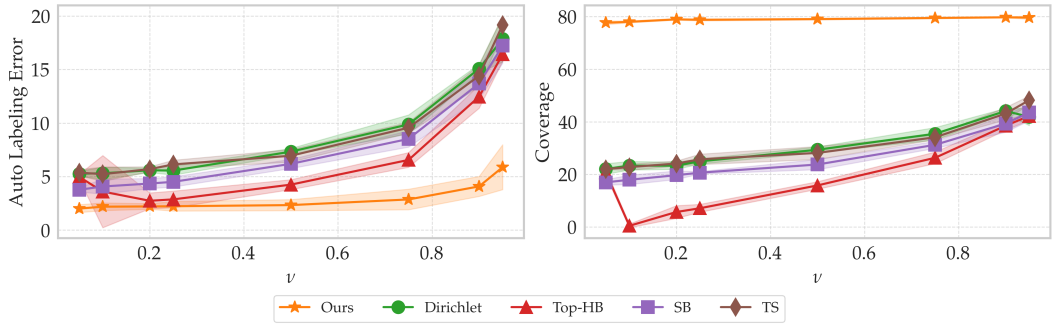


Figure 7: Autolabeling error and coverage of different post-hoc methods on CIFAR-10 for various ν

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 5 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 6, 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 7 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 3: 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 4: 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.2 Experiments on `Colander` input

Figure 14 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 3 and 4 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.

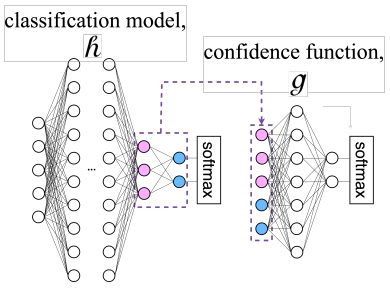


Figure 14: Our choice of g function.

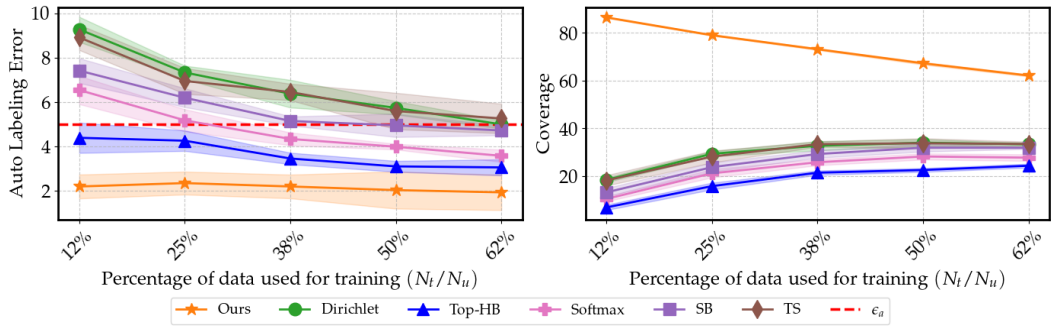


Figure 8: Auto-labeling error and coverage for different post-hoc methods on CIFAR-10 while we vary N_t. $N_u = 40,000$ is the size of the given unlabeled pool.

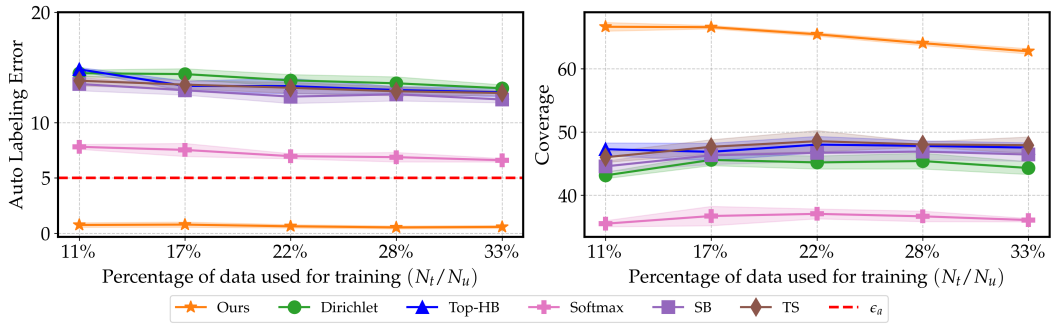


Figure 9: Auto-labeling error and coverage for different post-hoc methods on Tiny-ImageNet while we vary N_t. $N_u = 90,000$ is the size of the given unlabeled pool.

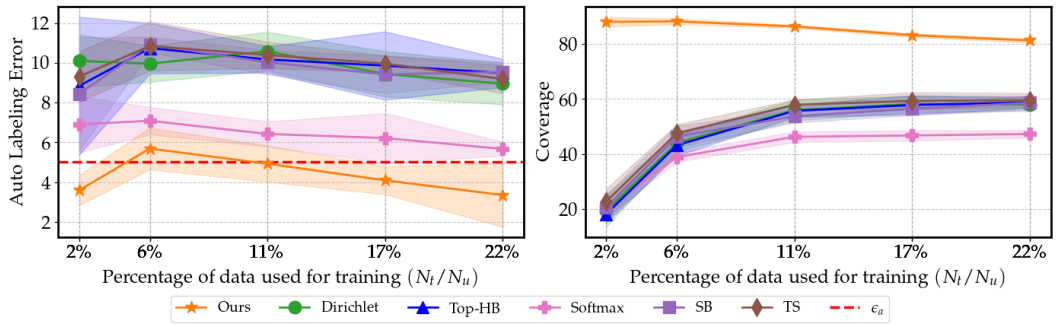


Figure 10: Auto-labeling error and coverage for different post-hoc methods on 20 Newsgroups while we vary N_t. $N_u = 9,052$ is the size of the given unlabeled pool.

C.3 Experiments on ϵ_a

We run TBAL with five values of $\epsilon_a \in \{0.01, 0.025, 0.05, 0.075, 0.1\}$ and report the results in Table 5. As expected the auto-labeling error is high with larger values of and smaller with small ϵ_a.

C.4 Experiments on multiple rounds

We further demonstrate that the performance gains are due to the use of Colander, even if methods use multiple rounds. To do so, we show the evolution of coverage and error over multiple rounds in Figure 15. The effects of using Colander are visible from the first round itself, and the following rounds improve performance further. We also run a single round (passive) variant of TBAL where we sample all the human-labeled points for training (N_t) randomly at once, train a classifier, do auto-labeling, and then stop. This setting avoids confounding due to multiple rounds. We observe that using Colander yields significantly higher coverage in comparison to the baselines (see Table 6).

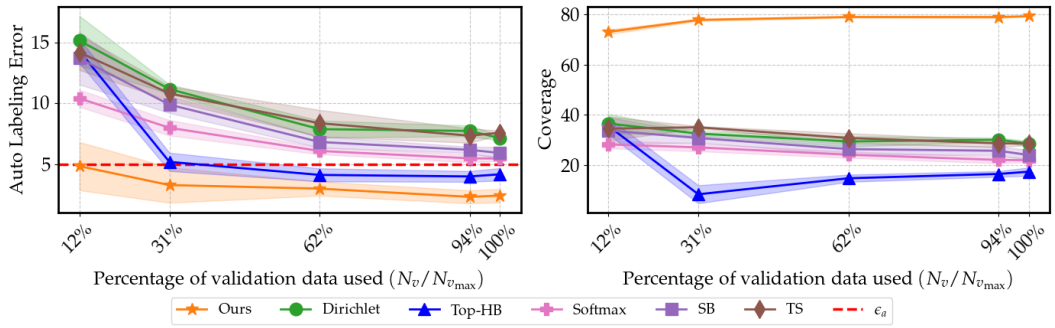


Figure 11: Auto-labeling error and coverage for different post-hoc methods on CIFAR-10 while we vary N_v. $N_{v_{\max}} = 8,000$ is the maximum number of points available for validation.

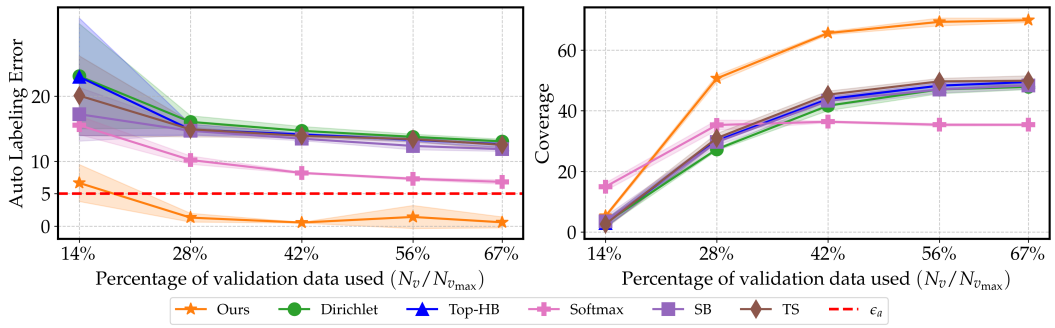


Figure 12: Auto-labeling error and coverage for different post-hoc methods on Tiny-ImageNet while we vary N_v. $N_{v_{\max}} = 18,000$ is the maximum number of points available for validation.

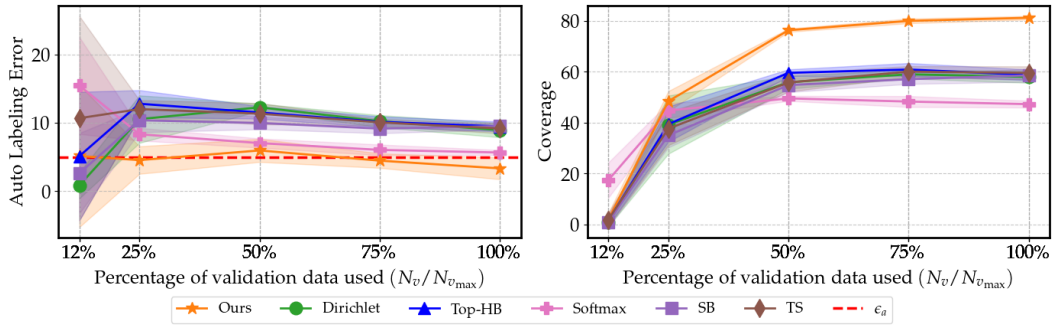


Figure 13: Auto-labeling error and coverage for different post-hoc methods on 20 Newsgroups while we vary N_v. $N_{v_{\max}} = 1,600$ is the maximum number of points available for validation.

This reinforces the fact that the gains in the multi-round TBAL are directly due to Colander, while multiple rounds of data selection, training, and auto-labeling are superior to doing everything in a single round.

Post-hoc Method	ϵ_a = 0.01		ϵ_a = 0.025		ϵ_a = 0.05		ϵ_a = 0.075		ϵ_a = 0.1	
	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)	Err (↓)	Cov (↑)
Softmax	5.86 ± 0.38	12.73 ± 1.61	5.86 ± 0.38	12.73 ± 1.61	4.78 ± 0.21	14.01 ± 2.08	6.80 ± 0.47	16.73 ± 1.19	9.03 ± 0.17	21.28 ± 0.82
TS	8.19 ± 0.88	19.44 ± 1.16	8.19 ± 0.88	19.44 ± 1.16	7.26 ± 0.29	23.15 ± 0.7	9.24 ± 0.78	22.49 ± 0.74	11.63 ± 0.51	25.79 ± 1.97
Dirichlet	8.22 ± 0.4	16.94 ± 1.2	8.22 ± 0.4	16.94 ± 1.2	7.6 ± 0.48	22.36 ± 1.18	9.68 ± 0.82	18.65 ± 0.97	11.26 ± 1.16	24.91 ± 2.09
SB	6.15 ± 0.52	11.74 ± 0.57	6.15 ± 0.52	11.74 ± 0.57	6.09 ± 0.35	18.58 ± 1.13	7.81 ± 0.65	17.37 ± 1.3	9.13 ± 1.08	20.52 ± 1.11
Top-HB	5.76 ± 0.42	9.89 ± 0.55	5.76 ± 0.42	9.89 ± 0.55	5.95 ± 0.7	15.58 ± 1.92	7.45 ± 0.8	13.84 ± 0.78	8.71 ± 1.37	17.9 ± 0.56
Ours	**1.2 ± 0.18**	**78.33 ± 0.76**	**1.32 ± 0.21**	**78.75 ± 0.4**	**2.96 ± 0.2**	**78.48 ± 0.17**	**4.3 ± 0.23**	**78.94 ± 0.42**	**6.29 ± 0.5**	**78.97 ± 0.46**

Table 5: ϵ_a **variation.** Dataset: CIFAR-10, Train-time method: Vanilla.

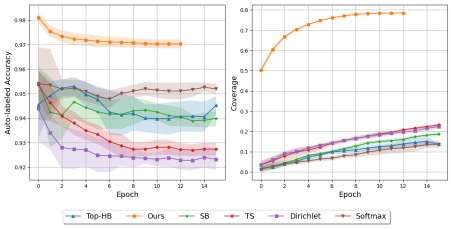


Figure 15: Clarification on multiple rounds. Per-epoch metrics for all post-hoc methods for CIFAR10. (left) Auto-labeling accuracy (right) Coverage. Train time method is vanilla and model is medium net.

Post-hoc method	Err (↓)	Coverage (↑)
Softmax	2.7 ± 0.54	11.06 ± 1.46
TS	3.04 ± 0.49	12.03 ± 1.98
Dirichlet	2.98 ± 0.32	11.22 ± 2.1
SB	2.72 ± 0.34	9.75 ± 1.33
Top-HB	1.83 ± 0.61	5.50 ± 1.08
Ours	**2.02 ± 0.28**	**49.62 ± 0.69**

Table 6: Results with single round of auto-labeling. Dataset and model: CIFAR-10 setting in the paper.

C.5 Experiments on different architectures

In TBAL it is not a priori clear what model the practitioner should use. The overall system is flexible enough to work with any chosen model class. Our focus is on evaluating the effect of various training time and post-hoc methods designed to improve the confidence functions for any given model. To answer the query, we ran experiments with Resnet18 and ViT models in the CIFAR-10 setting (see Table 7). As we expected there are variations in the results in the baselines due to model choices but our method maintains high performance irrespective of the classification model used. This is due to its ability to learn confidence scores tailored for TBAL.

Post-hoc Method	Err (↓)	Coverage (↑)
Softmax	14.02 ± 1.83	2.03 ± 0.31
TS	19.32 ± 2.51	2.54 ± 0.33
Dirichlet	17.27 ± 3.26	2.87 ± 0.55
SB	9.22 ± 10.91	0.46 ± 0.51
Top-HB	0.00 ± 0.00	0.00 ± 0.00
Ours	**2.62 ± 0.32**	**75.56 ± 0.15**

Post-hoc Method	Err (↓)	Coverage (↑)
Softmax	4.48 ± 0.23	33.24 ± 1.14
TS	6.38 ± 0.47	39.14 ± 1.96
Dirichlet	6.30 ± 0.41	37.99 ± 1.47
SB	5.16 ± 0.23	35.32 ± 1.36
Top-HB	4.46 ± 0.40	29.66 ± 0.74
Ours	**2.85 ± 0.25**	**78.56 ± 0.54**

Table 7: Model variation. CIFAR-10 dataset with ViT (Left) and ResNet18 (Right), Train-time method Vanilla.

C.6 Hyperparameters

The hyperparameters and their values we swept over are listed in Table 8 and 9 for train-time and post-hoc methods, respectively.

C.7 Train-time and post-hoc methods

C.7.1 Train-time methods

1. *Vanilla*: Neural networks are commonly trained by minimizing the cross entropy loss using stochastic gradient descent (SGD) with momentum [1, 3]. We refer to this as the Vanilla training method. We also include weight decay to mitigate the overconfidence issue associated with this method [10].

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 8: 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* [16]: 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)* [30]: 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 [15, 5]. 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* [53] aims to align confidence scores with the ordinal rankings criterion. It uses Sharpness Aware Minimizer (SAM) [6] to train the model, with the expectation that the flat minima would benefit the ordinal rankings objective of the confidence function.

C.7.2 Post-hoc methods

1. *Temperature scaling* [10]: This is a variant of Platt scaling [10], 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* [12]: 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 [52], this method focuses on calibrating the scores of predicted labels.

3. *Scaling-Binning* [24]: 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* [22]: 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 [52] has been a better choice over uniform width binning. Hence, we use uniform mass binning as well.

C.8 Compute resources

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

C.9 Detailed dataset and model

1. The MNIST dataset [26] consists of 28×28 grayscale images of hand-written digits across 10 classes. It was used alongside the LeNet5 [27], a convolutional neural network, for auto-labeling.
2. The CIFAR-10 dataset [20] contains $3 \times 32 \times 32$ color images across 10 classes. We utilized its raw pixel matrix in conjunction with SimpleCNN [17], a convolutional neural network with approximately 5.8M parameters, for auto-labeling.
3. Tiny-ImageNet [25] 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 [38] 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 [29, 35] is a natural language dataset comprising around 18,000 news posts across 20 topics. We used the FlagEmbedding [50] 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.10 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 8 and Table 9 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 9.

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 1.

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.

D Broader Impact

This paper contributes to the advancement of the practice of creating labeled datasets in machine learning. While our work has various possible societal implications, we do not identify any specific concerns that require special attention in this context.

E Related Work

Data Labeling. We briefly discuss prominent methods for labeling. Crowdsourcing [40, 43] 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 [9, 18, 28, 46, 45, 47, 4]. Weak supervision (WS), in contrast, emphasizes labeling through multiple inexpensive but noisy sources, not necessarily human [39, 7, 42, 48]. Works such as [39, 7] concentrate on binary or multi-class labeling, while [42, 48] extend WS 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 to obtain human labels while generating high-quality labeled data using a specific model. [37] use a TBAL-like algorithm and explore the cost of training for auto-labeling with large-scale model classes. Recent work [49] theoretically analyzes the sample complexity of validation data required to guarantee the quality of auto-labeled data.

Overconfidence and calibration. The issue of overconfidence [44, 33, 14, 2] is detrimental in several applications, including ours. Many solutions have emerged to mitigate the overconfidence and miscalibration problem. Gawlikowski et al. [8] provide a comprehensive survey on uncertainty quantification and calibration techniques for neural networks. Guo et al. [10] evaluated a variety of solutions ranging from the choice of network architecture, model capacity, weight decay regularization [21], histogram-binning and isotonic regression [51, 52] and temperature scaling [36, 34] 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 [23, 32, 31, 16]. On the other hand, post-hoc methods such as top-label histogram-binning [11], scaling binning [24], Dirichlet calibration [22] 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 [15]. To get such scores, Corbière et al. [5] add a module in the net for failure prediction, Zhu et al. [53] switch to sharpness aware minimization [6] to learn the model; CRL [30] regularizes the loss.

F NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims are backed by our novel technique in Section C and thorough empirical evaluation in Section 4.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: We discuss them briefly.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: It is an empirical paper, it does not have theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All the necessary details are provided in Section 4 and in the Appendix C. We have also uploaded the code along with the submission.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use publicly available datasets and uploaded the code as supplementary material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: These details are provided in the Section 4 and Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We run each setting with multiple random seeds and report the mean, standard deviations of the evaluation metrics.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Provided in the Appendix C.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have followed the NeurIPS Code of Ethics to the best of our knowledge.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper has a brief discussion on the broader impacts.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

Justification: The paper does not release such data or models that have high risk for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We have appropriately credited them along with citations.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The code is well documented along with instructions to run.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: We did not use crowdsourcing or human subjects in the paper.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: We did not use crowdsourcing or human subjects in the paper.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

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