Benchmarking Graph Conformal Prediction: Empirical Analysis, Scalability, and Theoretical Insights

Anonymous authors
Paper under double-blind review

Abstract

Conformal prediction has become increasingly popular for quantifying the uncertainty associated with machine learning models. Recent work in graph uncertainty quantification has built upon this approach for conformal graph prediction. The nascent nature of these explorations has led to conflicting choices for implementations, baselines, and method evaluation. In this work, we analyze the design choices made in the literature and discuss the tradeoffs associated with existing methods. Building on the existing implementations for existing methods, we introduce techniques to scale existing methods to large-scale graph datasets without sacrificing performance. Our theoretical and empirical results justify our recommendations for future scholarship in graph conformal prediction.

1 Introduction

Modern machine learning models trained on losses based on point predictions are prone to be overconfident in their predictions (Guo et al., 2017). The Conformal Prediction (CP) framework (Vovk et al., 2005) provides a mechanism for generating statistically sound post hoc prediction sets (or intervals, in case of continuous outcomes) with coverage guarantees under mild assumptions. The usual assumption made in CP is that data are exchangeable, i.e., the joint distribution of the data is invariant to permutations of the data points. CP's guarantees are distribution-free and can be added post hoc to arbitrary black-box predictor scores, making them ideal candidates for quantifying uncertainty in complex models, such as neural networks.

Network-structured data such as social networks, transportation networks, and biological networks are ubiquitous in modern data science applications. Graph Neural Networks (GNNs) have been developed to model vector representations of network-structured data and be effective in a variety of tasks such as node classification, link prediction, and graph classification (Hamilton, 2020; Wu et al., 2022). Uncertainty quantification approaches built for independent and identically distributed (iid) data cannot directly be applied to graph data, as the network structure introduces possible dependencies between the data points. However, recent work (Clarkson, 2023; Zargarbashi et al., 2023; Huang et al., 2023) has demonstrated that, in specific settings, CP can be applied to graph data to generate statistically sound prediction sets for the node classification task, similar to other classification tasks. These methods were specifically designed to leverage the graph structure and homophily to improve upon baseline non-graph CP methods. Variations of CP include full CP (Vovk et al., 2005), which has significant computational cost as the score function requires recomputation with replacement for each data point within the calibration set, cross-conformal prediction (Vovk, 2015)/CV+/Jackknife+ (Barber et al., 2021), and split (or inductive) conformal prediction (Papadopoulos et al., 2002; Papadopoulos, 2008). Prior work on graphs has mainly focused on the Split-CP setting due to its computational efficiency and distribution-free guarantees with black-box models, so we also focus on Split-CP for our work.

Key Contributions: This work makes the following contributions:

1. We perform a comprehensive analysis of the existing graph split-CP literature (e.g. CFGNN, DAPS) for node classification. This analysis aims to understand the choices made in method implementation (i.e., scalability) and method evaluation (i.e., dataset preparation, and baseline selection). We extend

this analysis to include more general and popular CP methods such as TPS and APS to observe when graph-specific methods improve.

2. We present a theorem for evaluating the efficiency (set size) of different conformal prediction methods *prior to* deployment.

3. We develop a Python library that implements our analysis, enabling practitioners to efficiently apply and evaluate conformal prediction for graph data, with scalability enhancements for large graphs, particularly for compute-intensive methods like CFGNN.

2 Conformal Prediction

Conformal prediction is used to quantify model uncertainty by providing prediction sets/intervals with rigorous coverage guarantees. We will focus on conformal prediction in the classification setting. Given a calibration dataset $\mathcal{D}_{\text{calib}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, where $\boldsymbol{x}_i \in \mathcal{X} = \mathbb{R}^d$ and $y_i \in \mathcal{Y} = \{1, \ldots, K\}$, conformal prediction can be used to construct a prediction set C for an unseen test point $(\boldsymbol{x}_{n+1}, y_{n+1})$ such that

$$\Pr\left[y_{n+1} \in C(\boldsymbol{x}_{n+1})\right] \geq 1 - \alpha$$

where $1 - \alpha \in (0, 1)$ is a user-specified coverage level. The only assumption required for the coverage guarantee is that $\mathcal{D}_{\text{calib}} \cup \{(\boldsymbol{x}_{n+1}, y_{n+1})\}$ is exchangeable. The following theorem provides a general recipe for constructing a prediction set with the coverage guarantee.

Theorem 2.1 ((Vovk et al., 2005; Angelopoulos & Bates, 2021))**.** *Suppose $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n+1}$ are exchangeable, $s : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a score function measuring the non-conformity of point $(\boldsymbol{x}, y)$, with higher scores indicating lower conformity, and a target $\alpha \in [0, 1]$. Let $\hat{q}(\alpha) = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha) \rceil}{n}; \{s(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}\right)$ be the conformal quantile. Define $C(\boldsymbol{x}) = \{y \in \mathcal{Y} : s(\boldsymbol{x}, y) \leq \hat{q}(\alpha)\}$. Then,*

$$1 - \alpha + \frac{1}{n+1} \geq \Pr\left[y_{n+1} \in C(\boldsymbol{x}_{n+1})\right] \geq 1 - \alpha \tag{1}$$

For the upper bound, we assume that the non-conformity scores are unique or there is a suitably randomly tie-breaking method.

The function s is the non-conformity score function, and it measures the degree of non-agreement between an input $\boldsymbol{x}$ and the label y, given exchangeability with the calibration data $\mathcal{D}_{\text{calib}}$, i.e., larger scores indicate worse agreement between $\boldsymbol{x}$ and y. While Theorem 2.1 does not restrict the choice of s, this choice can significantly impact the size of the prediction set.

The setup of theorem 2.1 is called Split-CP, as the score function remains fixed for the calibration split. In other versions of CP, the score function is usually more expensive as it requires some $1 < k \leq n$ calibration points in addition to the unseen test point. We have $k = n$ for full conformal prediction, whereas for methods such as cross-conformal prediction and CV+/Jackknife+, $k < n$. When applying Split-CP, the dataset is partitioned into $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{valid}} \cup \mathcal{D}_{\text{calib}} \cup \mathcal{D}_{\text{test}}$.

2.1 Conformal Prediction in Graphs

The usual tasks of interest with graph data include node classification, link prediction, and graph classification. This work focuses on node classification and its extensions to conformal prediction. Consider an attributed homogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\boldsymbol{X}$ is the set of node attributes. Let $\boldsymbol{A}$ denote the adjacency matrix for the graph. Further, let $\mathcal{Y} = \{1, \ldots, K\}$ denote the set of class labels associated with the nodes. For $v \in \mathcal{V}$, $\boldsymbol{x}_v \in \mathcal{X} = \mathbb{R}^d$ denotes its features and $y_v \in \mathcal{Y}$ denotes its true class label. The task of node classification is to learn a model that predicts the label for each node given node features and the adjacency matrix, i.e. $(\boldsymbol{X}, \boldsymbol{A}, v) \mapsto y_v$. Corresponding to the CP partitions, we denote the nodes in the training set as $\mathcal{V}_{\text{train}}$, the validation set as $\mathcal{V}_{\text{valid}}$, the calibration set as $\mathcal{V}_{\text{calib}}$, and the test set as $\mathcal{V}_{\text{test}}$. We denote $\mathcal{V}_{\text{dev}} = \mathcal{V}_{\text{train}} \cup \mathcal{V}_{\text{valid}}$ as the development set of the base model (non-conformalized). Note that labels are available only for nodes in $\mathcal{V}_{\text{train}}$, $\mathcal{V}_{\text{valid}}$, and $\mathcal{V}_{\text{calib}}$, and must be predicted for nodes in $\mathcal{V}_{\text{test}}$. The model cycle will involve four phases, viz. training, validation, calibration, and testing. Next, we discuss the different settings for node classification in graphs and the applicability of conformal prediction.

Table 1: Summary statistics for datasets. Predefined splits from the original sources are noted.

Dataset	Nodes	Edges	Classes	Features	# Train	# Valid	# Test
Amazon_Computers	13,752	491,722	10	767	-	-	-
Cora	19,793	126,842	70	8,710	-	-	-
Coauthor_CS	18,333	163,788	15	6,805	-	-	-
Flickr	89,250	899,756	7	500	44,625	22,312	22,313
ogbn-arxiv	169,343	1,166,243	40	128	90,941	29,799	48,603

Transductive setting In this setting, the model has access to the fixed graph $\mathcal{G}$ during the model cycle. The nodes used in the model cycle are split into $\mathcal{V}_{\text{train}}$, $\mathcal{V}_{\text{valid}}$, and $\mathcal{V}_{\text{calib}} \cup \mathcal{V}_{\text{test}}$. The specific $\mathcal{V}_{\text{calib}}$ and $\mathcal{V}_{\text{test}}$ are subsets of $\mathcal{V}_{\text{calib}} \cup \mathcal{V}_{\text{test}}$. This is the setting considered by Zargarbashi et al. (2023) and Huang et al. (2023). Note that the labels for $\mathcal{V}_{\text{calib}}$ are not available for training and validation of the base model; however, all the neighborhood information of $\mathcal{G}$ and the features $\boldsymbol{x}_v$ and labels y_v, for $v \in \mathcal{V}_{\text{dev}}$ are available. During the calibration phase, the $(\boldsymbol{x}_v, y_v)$ for $v \in \mathcal{V}_{\text{calib}}$ and all the neighborhood information are used to compute the non-conformity scores. This split ensures that the base model cannot distinguish between the calibration and test nodes, and hence exchangeability holds for $v \in \mathcal{V}_{\text{calib}} \cup \mathcal{V}_{\text{test}}$.

Inductive setting In this setting, the model is only trained and calibrated on a fixed graph $\mathcal{G}$ during the model cycle. Once the calibration phase is complete, $\mathcal{G}$ changes to include new unseen test points, breaking the exchangeability assumption as nodes arriving later in the sequence will have access to neighbors that came earlier. This setting is infrequently studied as it requires making specific assumptions of graph structure and the data generation process.

In line with previous work, we focus on the ***transductive*** setting. The following theorem states that a scoring model trained on the calibration set will generate scores exchangeable with the test set in the transductive setting, thus allowing for conformal prediction.

Theorem 2.2 ((Zargarbashi et al., 2023; Huang et al., 2023))**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ be an attributed graph, and $\mathcal{V}_{calib} \cup \mathcal{V}_{test}$ be exchangeable. Let $F : \mathcal{X}^{|V|} \to \Delta^{|V| \times K}$ be any permutation equivariant model on the graph (e.g., GNNs). Define $F(G) = \Pi \in \Delta^{|V| \times K}$ as the output probability matrix for a model trained on only $\mathcal{V}_d$. Then any score function $s(v, y) = s(\Pi_v, y, \mathcal{G})$ is exchangeable for all $v \in \mathcal{V}_{calib} \cup \mathcal{V}_{test}$.*

The intuition for this theorem is that if the underlying model does not depend on the order of the nodes in the graph, then the outputs will also be exchangeable. This holds for most standard GNNs. The formal proof for this theorem is available in (Zargarbashi et al., 2023; Huang et al., 2023). This theorem paves the way for using conformal prediction for transductive node classification.

For the following sections, we will assume that the base model $\hat{\pi} : \mathcal{X} \to \Delta_{\mathcal{Y}}$, where $\Delta_{\mathcal{Y}}$ is the probability simplex over $\mathcal{Y}$, is learned using the training and validation sets $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{valid}}$. The calibration set $\mathcal{D}_{\text{calib}}$ is used to determine the $\hat{q}(\alpha)$ from Theorem 2.1 and the test set $\mathcal{D}_{\text{test}}$ is the set for which we want to construct our prediction sets. Note that the scores need not lie over a simplex, but instead be in $\mathbb{R}^K$. However, this greatly simplifies the exposition for the following sections and is standard practice in prior work.

3 Empirical Analysis and Insights

Datasets and Methods Table 1 contains a representative set of datasets of varying sizes (i.e., number of nodes/edges) and the number of classes evaluated in this section. The Appendix contains the list of all the datasets used in this study. We used the dataset versions available on the Deep Graph Library (Wang et al., 2019) and Open Graph Benchmark (Hu et al., 2020).

We compare CP methods, including TPS (Sadinle et al., 2019), APS (Romano et al., 2020), and RAPS (Angelopoulos et al., 2021) and describe the setups appropriate for their use in graph settings. For graph-focused CP methods, we include CFGNN (Huang et al., 2023), DAPS (Zargarbashi et al., 2023), and NAPS (Clarkson, 2023). To adapt NAPS to the transductive setting, we build on Zargarbashi et al. (2023) by using APS as a

baseline score and compute a weighted quantile of scores from the set of k-hop nodes - of a given test node - that intersect the calibration nodes. (Further discussion in Appendix).

Metrics For evaluation, we used the following standard metrics (Shafer & Vovk, 2008):

(i) **Coverage** is the proportion of test points where the true label is in the prediction set, i.e., $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \mathcal{D}_{\text{test}}} \mathbf{1}[y_i \in C(\boldsymbol{x}_i)]$.

(ii) **Efficiency** is the average prediction set size, i.e., $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \mathcal{D}_{\text{test}}} |C(\boldsymbol{x}_i)|$

(iii) **Label (or Class) Stratified Coverage** (Sadinle et al., 2019) is similar to Coverage, but is concerned with achieving the following conditional guarantee: $\Pr(y_{n+1} \in C(\boldsymbol{x}) \mid y_{n+1} = y) \geq 1-\alpha,\ \forall y \in \mathcal{Y}$. As a metric, it is defined as the mean of the coverage for each class, i.e., $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \mathcal{D}_{\text{test}}} \left(\frac{1}{K} \sum_{k=1}^{K} \mathbf{1}[y_i \in C(\boldsymbol{x}_i), y_i = k] \right)$.

Dataset Splits and Training. There are several methods of partitioning $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$, $\mathcal{D}_{\text{calib}}$, and $\mathcal{D}_{\text{test}}$. Two methods used in existing works on graph conformal prediction for node classification are

(1) **Full-Split (FS) Partitioning** (Huang et al., 2023). In this splitting scheme, the data is split such that each subset of the partition adheres to a size constraint based on $\mathcal{D}$. For example, in CFGNN (Huang et al., 2023) the authors split the datasets in their experiments randomly, satisfying a 20%/10%/35%/35% split into $\mathcal{D}_{\text{train}}/\mathcal{D}_{\text{valid}}/\mathcal{D}_{\text{calib}}/\mathcal{D}_{\text{test}}$. Note that the overall percentage of data for which labels are provided (in either the development or calibration set) is a large proportion (65%) of the full dataset. This splitting scheme is ideal for non-conformity score models with numerous trainable (or tunable) parameters as more calibration data is available for training. We explore the following splitting schemes under FS partitioning: $(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \mathcal{D}_{\text{calib}}, \mathcal{D}_{\text{test}}) = (0.2, 0.1, 0.35, 0.35)$, $(0.2, 0.2, 0.3, 0.3)$, $(0.3, 0.1, 0.3, 0.3)$, and $(0.3, 0.2, 0.25, 0.25)$.

(2) **Label-Count (LC) Sample Partitioning** (Zargarbashi et al., 2023). In this splitting scheme, the data is split to ensure an equal number of samples for each class label is present in $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$, and $\mathcal{D}_{\text{calib}}$. The remaining nodes are $\mathcal{D}_{\text{test}}$. This setting is common when only a small proportion of training/labeled nodes are available (e.g., semi-supervised learning). Intuitively, this setting is ideal for methods that do not have many parameters to train. We explore setting the number of samples per class to 10, 20, 40, and 80. Note that we assign nodes of each class sequentially, so it is feasible in this setup to have some classes with no representative samples in some data subset. If the dataset has **predefined splits** (e.g., Flickr, ogbn-arxiv), in addition to the described splitting rules, we ensure that $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$ come solely from the training and validation splits, while $\mathcal{D}_{\text{calib}} \cup \mathcal{D}_{\text{test}}$ come from the test split. Finally, we note that this is a biased sampling scheme which may violate the exchangeability assumption; however, we include this scheme as it is used in the literature Zargarbashi et al. (2023; 2024) for completeness of our analysis.

3.1. On TPS and adaptability. Threshold Prediction Sets (TPS) (Sadinle et al., 2019) is a simple technique for generating conformal prediction sets. The score function $s(\boldsymbol{x}, y) = 1 - \hat{\pi}(\boldsymbol{x})_y$ directly maps the probability from the base model for the correct class into a non-conformity score. The score is higher if the model outputs a lower probability for the correct class, indicating the label conforms less with the model. A $1-\alpha$ (approximate) quantile creates a probability inclusion threshold for this score over the calibration set, ensures coverage, and can be shown to generate prediction sets with the best-expected efficiency (Sadinle et al., 2019). However, TPS is known to achieve this efficiency by undercovering hard examples and overcovering easy examples (Angelopoulos et al., 2021; Zargarbashi et al., 2023). By overcovering easy examples, TPS still maintains the overall coverage guarantee without correctly accounting for the coverage of the harder examples.

We note that this discrepancy is claimed to occur as the TPS scores are not 'adaptive', and consider only one dimension of the score for each calibration sample. However, Sadinle et al. (2019) also proposed a classwise control version of TPS. Instead of defining a single threshold for all classes, they separately compute the

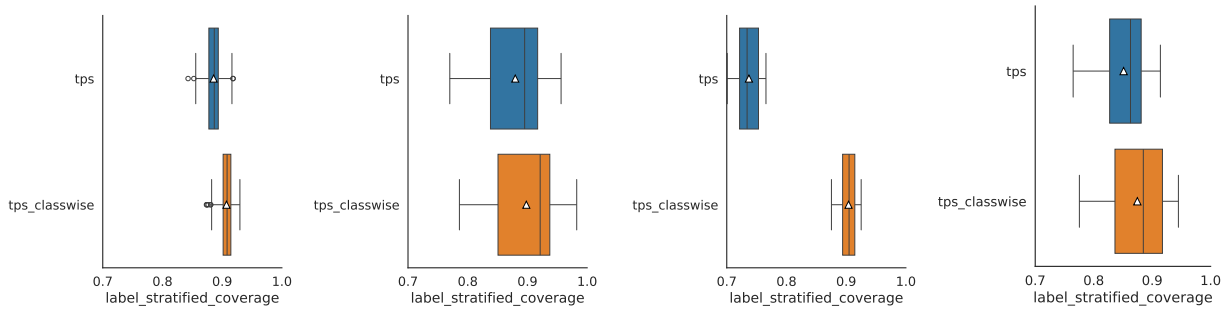
threshold for each class for a corresponding α. Thus, we define classwise quantile thresholds for each $y \in \mathcal{Y}$

$$\hat{q}(\alpha, y) = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha) \rceil}{n}; \{s(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, n, y_i = y\}\right) \tag{2}$$

and the corresponding prediction sets as

$$C_{\text{TPS}}(\boldsymbol{x}) = \{y \in \mathcal{Y} : s(\boldsymbol{x}, y) \leq \hat{q}(\alpha, y)\}$$

Note that this version provides coverage for each class label, making it more 'adaptive' (i.e., it achieves label stratified coverage guarantee). The version defined by Sadinle et al. (2019) allows controlling α_y for each class, though we set $\alpha_y = \alpha$ for class-adaptability. The trade-off with the adaptive version is that we have fewer calibration samples for each quantile threshold dimension, which may lead to higher variance in the distribution of coverage (Vovk, 2012). We call this variation of TPS, TPS-Classwise, and consider it in our baselines for comparison. From Figure 1, we see that using TPS-Classwise successfully provides label-stratified coverage in both the FS and LC split settings.



(a) Amazon_Computers (b) ogbn-arxiv

Figure 1: We set the target coverage rate $\alpha = 0.1$. The boxplots present the Label Stratified Coverage for (a) Amazon_Computers and (b) ogbn-arxiv for both the FS split (left) and LC split (right). We want the means (white triangle) to be around $1 - \alpha = 0.9$. For Labeled Stratified Coverage, TPS-Classwise is comparable to or better than TPS.

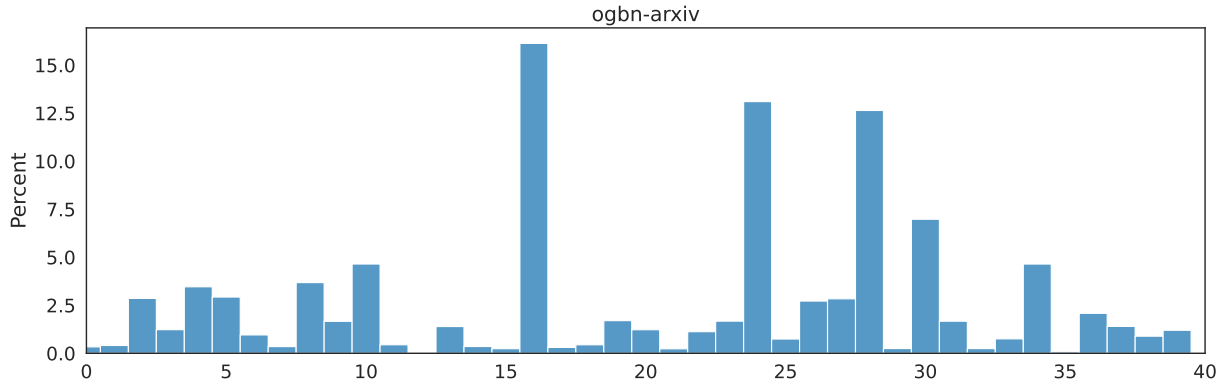


Figure 2: Label Distribution for ogbn-arxiv

3.2. On APS and randomization. The most popular baseline in the (graph) conformal prediction literature is Adaptive Prediction Sets (APS). Romano et al. (2020) introduce APS by defining an optimal prediction set construction mechanism by first considering an oracle-specified probability and then generalizing it to predictive probabilities. Consider a probability prediction function that estimates $\widehat{\Pr}[Y = y | X_{test} = \boldsymbol{x}] =$

$\hat{\pi}_y(\boldsymbol{x})$ for each $y \in \mathcal{Y} = \{1, \ldots, K\}$. Assume that $\hat{\pi}$ are all distinct – for ease of defining rank. Suppose the rank of the true class amongst the sorted $\hat{\pi}$ is r_y, i.e., $\sum_{i=1}^{K} \mathbf{1}[\hat{\pi}_i(\boldsymbol{x}) \geq \hat{\pi}_y] = r_y$. Following the APS definitions, considering a uniform random variable $u \sim U(0, 1)$, it is possible to derive a randomized non-conformity score (see Appendix A, for derivations),

$$A(\boldsymbol{x}, y, u; \hat{\pi}) = \left[\sum_{i=1}^{r_y} \hat{\pi}_{(i)}(\boldsymbol{x}) \right] - u\hat{\pi}_y \tag{3}$$

Instead, if a deterministic approach is used to define the conformal score instead (i.e., the randomized term is not included), then we could add the probabilities until the true class is included:

$$\tilde{A}(\boldsymbol{x}, y; \hat{\pi}) = \left[\sum_{i=1}^{r_y} \hat{\pi}_{(i)}(\boldsymbol{x}) \right] \tag{4}$$

The version of APS without randomization still provides the same conditional coverage guarantees with a simpler exposition since prediction sets are constructed by greedily including the classes until the true label is included. Thus, this version is implemented in the popular monographs on conformal prediction by Angelopoulos & Bates (2021). However, the lack of randomization may sacrifice (prediction) efficiency. This modification affects computing the conformal quantile during the calibration phase and the prediction set construction during the test phase.

We will now formally present the conditions that impact the efficiency. Let $A(\boldsymbol{x}, y)$ be any non-conformity score function, and,

$$\hat{q}_A = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha) \rceil}{n}; \{A(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} \right)$$

Define $A_i(y) := A(\boldsymbol{x}_i, y)$ where $y \in \mathcal{Y}$. Recall that the prediction set for a point $\boldsymbol{x}_i$ using A as the non-conformity score is

$$C_A(\boldsymbol{x}_i) = \{y \in \mathcal{Y} : A_{n+1}(y) \leq \hat{q}_A\}.$$

Define $C_A^i = C_A(\boldsymbol{x}_i)$ for brevity. Let $\mathcal{Y}_i' = \mathcal{Y} \setminus \{y_i\}$ be the set of incorrect labels for $\boldsymbol{x}_i$ and $y_i' \in \mathcal{Y}_i'$ be any incorrect class label for each $\boldsymbol{x}_i$.

Now, recall $(\boldsymbol{x}_i, y_i)_{i=1}^{n+1}$ are assumed to be exchangeable. Thus we note that $(\boldsymbol{x}_i, \mathcal{Y}_i')_{i=1}^{n+1}$ must also be exchangeable since $\mathcal{Y}_i'$ is the complement of $\{y_i\}$. Lastly, consider an IID sequence $u_i \sim U(\{1, \ldots, K-1\})$ for $i = 1, \ldots, n+1$. Since $\{u_i\}_{i=1}^{n+1}$ is IID, we can adjoin it to the covariates and produce an exchangeable sequence of triples $(\boldsymbol{x}_i, \mathcal{Y}_i', u_i)_{i=1}^{n+1}$.

From this exchangeable sequence, we can compute the non-conformity scores, and $(\{A(\boldsymbol{x}_i, y_i') \mid y_i' \in \mathcal{Y}_i'\}, u_i)_{i=1}^{n+1}$. Lastly, to get a pointwise representative for each set, let $y_i^R = y_{i(u_i)}'$ such that $A(\boldsymbol{x}_i, y_{i(1)}') \leq A(\boldsymbol{x}_i, y_{i(2)}') \leq \cdots \leq A(\boldsymbol{x}_i, y_{i(K-1)}')$ – assuming there is a suitably random tie-breaking method for identical scores. Then, $\{A(\boldsymbol{x}_i, y_i^R)\}_{i=1}^{n+1}$ is an exchangeable sequence. In practice, the process of ordering the $\mathcal{Y}_i'$ labels is not necessary. However, it concretely demonstrates the procedure to form $\{A(\boldsymbol{x}_i, y_i^R)\}_{i=1}^{n+1}$ maintains exchangeability.

We now define $\alpha_c^A \in [0, 1]$ such that:

$$\hat{q}_A = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha_c^A) \rceil}{n};\ \{A(\boldsymbol{x}_i, y_i^R)\}_{i=1}^{n} \right). \tag{5}$$

In other words, α_c^A is the miscoverage level for the random incorrect label when using $\hat{q}_A$ to construct the prediction set. To compute α_c^A, we utilize the following lemma[1].

[1]For full exposition we copy the proof of this lemma in Appendix B

Lemma 3.1 (Anonymous (2025)). *For $\lambda \in [0,1]$ and $n = |\mathcal{D}_{\text{calib}}|$,*

$$\frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) > \lambda]}{n+1} \leq \Pr[y_{n+1} \notin \mathcal{C}_\lambda(\boldsymbol{x}_{n+1})] \leq \frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) > \lambda] + 1}{n+1} \tag{6}$$

With Lemma 3.1 and setting $\lambda = \hat{q}_A$, we get $\alpha_c^A = \frac{\sum_{i=1}^{n} \mathbf{1}[A(\boldsymbol{x}_i, y_i^R) > \hat{q}_A] + 1}{n+1}$, which gives us

$$1 - \alpha_c^A \leq \Pr[y_{n+1}^R \in C_A^{n+1}] \leq 1 - \alpha_c^A + \frac{1}{n+1}. \tag{7}$$

Let $\tilde{A}(\boldsymbol{x}, y)$ be another non-conformity score and define $\{\tilde{A}(\boldsymbol{x}_i, y_i^R)\}_{i=1}^n$ and $\alpha_c^{\tilde{A}}$ similarly. The following theorem gives a sufficient condition for when A is more efficient than $\tilde{A}$.

Theorem 3.1. *If $\alpha_c^A - \alpha_c^{\tilde{A}} \geq \frac{2}{(n+1)}$ then score function A produces a more efficient prediction set than $\tilde{A}$. Formally, $\mathbb{E}\left[|C_{\tilde{A}}(\boldsymbol{x}_{n+1})| - |C_A(\boldsymbol{x}_{n+1})|\right] \geq 0$*

With Theorem 3.1, we can further determine the expected efficiency improvement as the calibration set size gets arbitrarily large as seen in the following corollary. Both proofs are available in Appendix B.

Corollary 3.1.1. *As $n \to \infty$, $\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_A^{n+1}|\right] = (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}}\right)$*

Applying Theorem 3.1 to randomized APS (A) and deterministic APS ($\tilde{A}$), intuitively, as each score in A gets shifted by a small $u\pi$ term to the left, q_A would be lower than $q_{\tilde{A}}$. Thus, the miscoverage levels we would search for in the complementary scores $1-\alpha_c^A$ would be less than $1-\alpha_c^{\tilde{A}}$. $1-\alpha_c^A < 1-\alpha_c^{\tilde{A}} \implies \alpha_c^A - \alpha_c^{\tilde{A}} > 0$. If the shift is sufficiently large, then the randomized prediction set will be more efficient than the non-randomized one. In Figure 3, we show what this looks like for a practical example over the Cora dataset. In Figure 3 (right), the normalized sorted index at which the lower threshold q_A is reached when considering the incorrect classes is lower, i.e., $1-\alpha_c^A$ is lower, and hence α_c^A is higher. As a part of the proof, we show dependencies on $\frac{1}{(n+1)}$ and $(K-1)$, which indicates the improvements are more pronounced for larger $\mathcal{D}_{\text{calib}}$ and many classes.

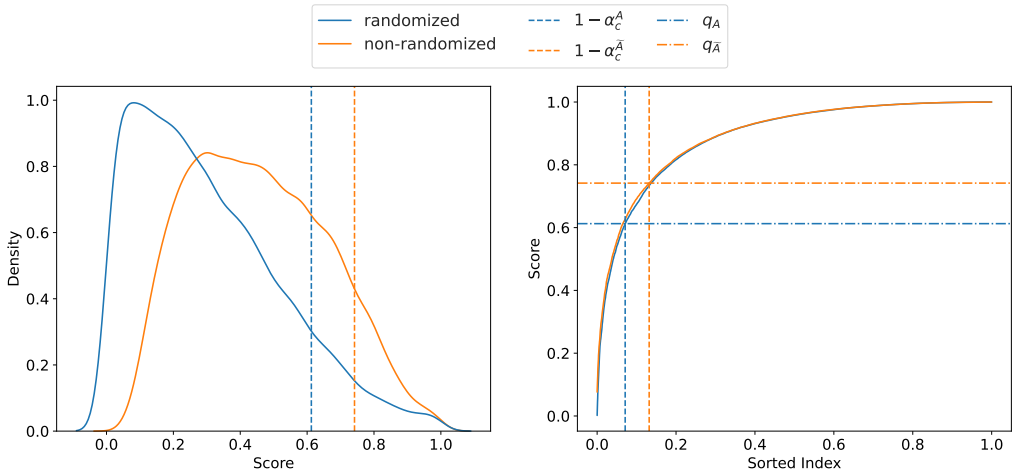


Figure 3: Scores for the Cora dataset using the randomized and non-randomized versions of APS. In the left plot, the vertical lines show the shift in the standard conformal quantiles for A (randomized APS) and $\tilde{A}$ (non-randomized APS) for 0.9 coverage. In the right plot, the vertical lines show the shift in the $1-\alpha_c$ value for A and $\tilde{A}$ using scores for the incorrect classes. We have $(1-\alpha_c^{\tilde{A}}) - (1-\alpha_c^A) \gg \frac{2}{n+1} \iff \alpha_c^A - \alpha_c^{\tilde{A}} \gg \frac{2}{n+1}$ which satisfies the condition for Theorem 3.1. Thus, A is more efficient as seen in the left plot since $q_A < q_{\tilde{A}}$.

Figure 4 provides box plots that compare the efficiency of randomized and non-randomized versions of APS across different datasets. We observe that for each split type,the randomized version consistently provides a

more efficient prediction set. This effect is most pronounced for datasets with many potential classes in the FS split, which aligns with the intuition from Theorem 3.1 described above. Overall, the empirical results show that the effect of randomized APS is more pronounced for larger values of K.

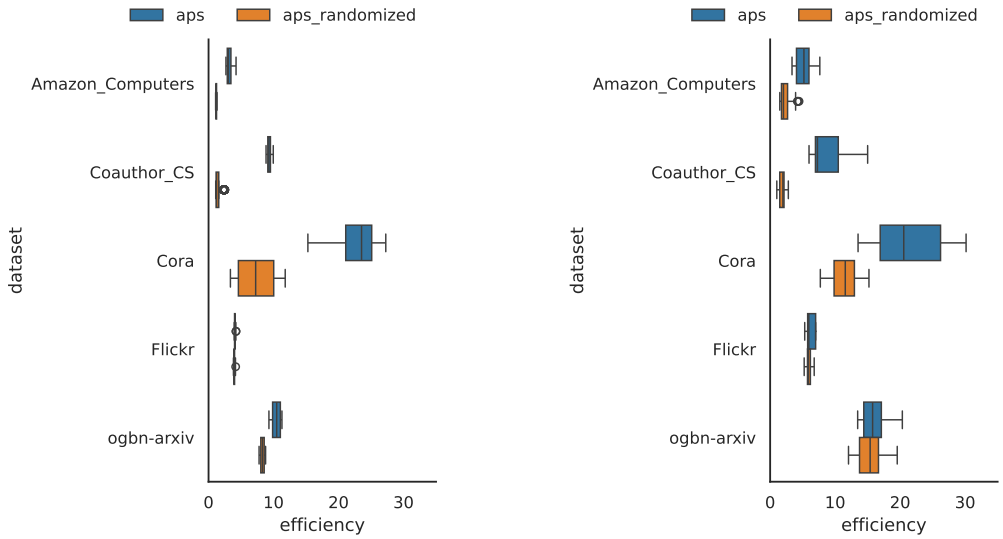


Figure 4: We set the target coverage rate $\alpha = 0.1$. Box plots depicting the efficiencies of APS and Randomized APS across different datasets and multiple runs in both the FS split (left) and LC split (right). Using randomization (the lower box plot for each dataset) consistently improves over the non-randomized version as the efficiencies are distributed around smaller values.

3.3. Conformalized GNN. CFGNN (Huang et al., 2023) is a recent GNN-based approach for conformal prediction. The underlying observation of this approach is that inefficiencies are correlated between nodes having similar neighborhood topology. With this intuition, during the calibration stage, a second GNN is trained to correct the scores from the base model while optimizing for the efficiency of the prediction sets. This is feasible as all the steps of the conformal prediction framework (i.e., non-conformity score computation, quantile computation, thresholding) can be expressed as differentiable operations (Stutz et al., 2021). Thus, the second GNN can be trained using an efficiency-based loss function, which Huang et al. (2023) proposes. More details on CFGNN are available in the Appendix.

Impact of Inefficiency Loss. The choice of conformal loss during calibration and test plays a vital role in determining the overall performance of GNN-based conformal prediction. To illustrate, we replicate an experiment by Huang et al. (2023) who use TPS for its inefficiency loss during the calibration stage and non-randomized APS when constructing the final prediction sets and show a significant improvement in efficiency over the baseline (Figure 5 right). However, if instead randomized APS loss is used (Figure 5 left), we observe that the baseline is competitive across various coverage thresholds. It is worth noting that CFGNN appears robust to this choice, although the gains in efficiency are not as dramatic in the randomized setting. We also note that the confidence bars are narrower in the randomized setting. Similar results were observed on other datasets (see Appendix E).

Based on these insights, we implemented an improved version of CFGNN, which uses APS with randomization for *both* training and evaluation, labeled as 'cfgnn_aps.' The original implementation is labeled as 'cfgnn_orig'. Our library implementation of CFGNN allows for either TPS or APS to be used for training and evaluation and is extensible to other conformal prediction methods.

We compare the efficiency of 'cfgnn_aps,' 'cfgnn_orig,' and 'aps_randomized' in Figure 6. Note that 'cfgnn_aps' improves upon or matches the efficiency of 'cfgnn_orig' and can even improve upon 'aps_randomized.' Originally CFGNN was only evaluated using FS splits, so we also benchmark CFGNN

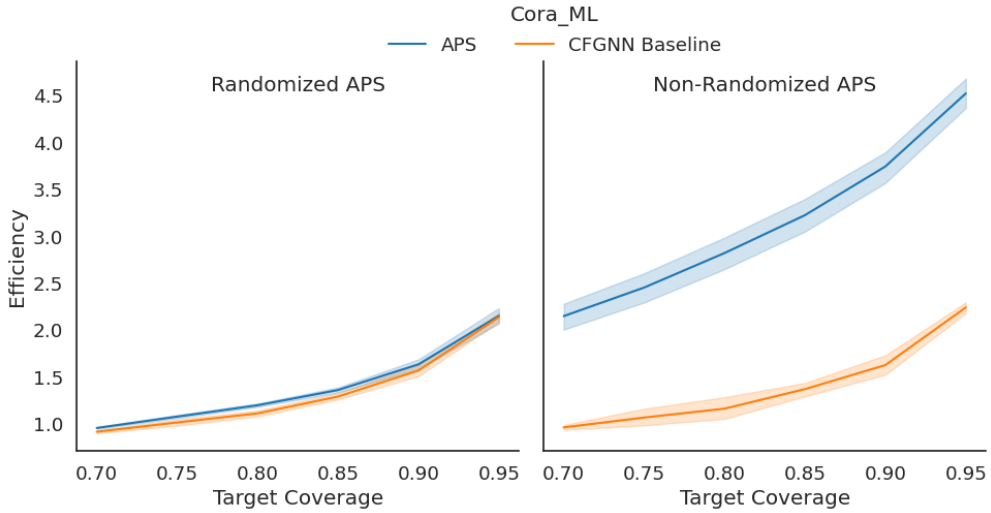


Figure 5: The plot on the right replicates an experiment (Huang et al., 2023) plotting efficiency over various coverage rates for the Cora_ML dataset (a subset of the Cora dataset) for both CFGNN and a baseline model. The plot on the left uses APS with randomization when constructing the final prediction sets. These plots illustrate the benefits of using randomization on baseline performance.

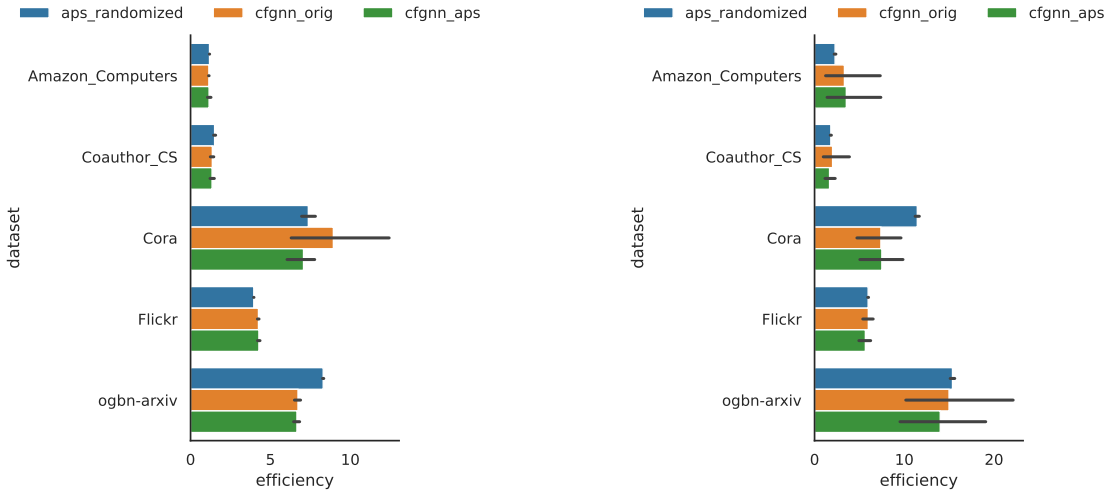


Figure 6: Bar charts denoting efficiency for 'cfgnn_aps', 'cfgnn_orig', and 'aps_randomized' for the FS split (left) and LC split (right) at $\alpha = 0.1$. We see that 'cfgnn_aps' improves or matches efficiency in most cases.

using LC splits in Figure 6. In the LC split setting, CFGNN is seen to be quite brittle. One potential reason is that the LC split style doesn't provide enough data to train the second GNN. An area of future work is to adapt CFGNN for the LC split setting.

Scaling CFGNN. In the original CFGNN implementation, full batch training was used. While this approach has merits, it also poses challenges, particularly when dealing with larger graphs. The need for a more scalable solution was evident, leading us to the modifications we have implemented. We implemented a batched version of CFGNN to ensure it can be used for larger graphs (e.g., ogbn-arxiv). To scale CFGNN for even larger graphs, we cache the outputs from the base model to be treated as features for the CFGNN training rather than having to sample neighbors for both the base model and CFGNN, significantly speeding up the computation in both training and evaluation for CFGNN.

Table 2: Impact of different CFGNN implementations starting from the baseline, then batching, and then caching and batching combined. Used the **best** CFGNN architecture (w.r.t. validation efficiency) for each dataset. We run 5 trials for each setup and report a 95% confidence interval.

(a) Impact on Runtime

dataset(↓) / method(→)	original	batching	batching+caching
Amazon_Computers	664.98 ± 10.90	205.29 ± 3.96	73.85 ± 1.56
Cora	1378.01 ± 13.35	203.61 ± 2.52	73.60 ± 1.53
Coauthor_CS	638.56 ± 6.14	88.49 ± 1.14	31.67 ± 0.53
Flickr	868.87 ± 9.98	567.39 ± 6.50	56.71 ± 1.30
ogbn-arxiv	410.91 ± 8.29	373.19 ± 3.64	111.38 ± 1.95

(b) Impact on Efficiency

dataset(↓) / method(→)	baseline	batching	batching+caching
Amazon_Computers	1.29 ± 0.05	1.15 ± 0.01	1.14 ± 0.01
Cora	6.81 ± 1.07	8.34 ± 1.12	7.96 ± 0.59
Coauthor_CS	1.10 ± 0.01	1.15 ± 0.01	1.14 ± 0.01
Flickr	4.23 ± 0.07	4.23 ± 0.04	4.24 ± 0.03
ogbn-arxiv	7.07 ± 0.05	7.28 ± 0.06	6.91 ± 0.01

We compare three variations of the CFGNN implementation to demonstrate the impact of batching and caching on the runtime. Across all comparisons, we use the FS split, with 20%/20% assigned to train/valid sets and 35% to the calibration dataset. We use the best base GNN (w.r.t accuracy) and best CFGNN (w.r.t efficiency) architecture determined via hyperparameter tuning.

The baseline implementation follows the setup by Huang et al. (2023), where the CFGNN is trained with full batch gradient descent for 1000 epochs. Our improved implementation, which uses mini-batch gradient descent, achieves comparable efficiency in only 50 epochs without any batch size tuning (we set the batch size to 64 for consistent comparison) as shown in Table 2b. Finally, we cache the output probabilities from the base GNN to the batched implementation, further reducing the runtime. Table 2 compares the batching and the combined batching + caching improvements. Our implementation can achieve improvements ranging from 3.69× (ogbn-arxiv) to 20.16× (Coauthor_CS) in runtime over the baseline implementation. [2]

3.4. Summary of Key Insights For Graph Conformal Prediction. We analyze several graph and non-graph CP methods proposed in the recent literature for transductive node classification. Based on our theoretical, empirical, and methodological results, we encourage practitioners to be precise and careful when developing and evaluating conformal prediction methods for graphs. We analyze the efficiency of all the methods that we consider in different datasets. Some of the key insights we report include:

- **TPS is Efficient.** TPS is consistently the most efficient method for each dataset, regardless of the train/validation/calibration split. However, this often comes at a cost to adaptability, as shown in Figure 7 for the Flickr dataset.
- **TPS-Classwise is a viable baseline.** While TPS is provably optimal for efficiency, the classwise version has largely been ignored in the literature as a baseline. Our results demonstrate that TPS-Classwise achieves label-stratified coverage–stemming from its classwise quantile procedure– making it an 'adaptive' method that is a competitive baseline. This adaptability usually comes at a small cost to efficiency, though this is not the case with all datasets (see PubMed in Figure E7).
- **Diffused Classwise TPS.** We also explored an alternative to classwise TPS where we apply the diffusion operator from Zargarbashi et al. (2023) on top of the 'adaptive' TPS-Classwise, discussed earlier. As with

[2]For each dataset and split type, we provide the configuration files for the best corresponding base GNN and CFGNN architectures in the attached code.

other datasets, DTPS also provides label-stratified coverage for Flickr (see Figure 7). More discussions on DTPS and Diffusion Adaptive Sets (DAPS) (Zargarbashi et al., 2023) can be found in the Appendix.

- **APS with randomization is provably more efficient than APS without randomization and should be the default setting for future evaluations involving APS.** From Theorem 3.1, we observe that the randomization term in APS Randomized decreases the incorrect label scores *substantially less* than the conformal quantile meaning these incorrect labels are less likely to be included when constructing the prediction set, therefore improving efficiency. This result is further pronounced for datasets with many classes because most incorrect labels shift less than the correct label, suggesting that APS Randomized should be a default baseline in the future.
- **NAPS provides a balanced tradeoff (and applies to the non-exchangeable setting).** A balance between efficiency and label-stratified coverage is achieved using the NAPS (Clarkson, 2023) scoring function. Before computing a quantile, NAPS uses neighbor distances to assign weights to non-conformity scores. In the original paper and our implementation, NAPS has a hyperparameter k, which is the maximum distance considered when assigning a weight, and everything beyond that is given zero weight. A unique characteristic of NAPS is that it is the only method discussed that provides guarantees for the non-exchangeable setting. More details on NAPS and its variations can be found in Appendix C. Appendix E has similar plots for the other datasets.
- **Randomization in CFGNN.** As noted in Figure 6, randomization within the inefficiency loss function can improve the efficiency of the original CFGNN implementation, particularly for datasets with many classes (e.g., Cora). It is also worth reiterating that the perceived efficiency gains of the original CFGNN are not as significant when compared to APS with randomization (see Figure 5).
- **Scaling up CFGNN.** We reimplement CFGNN using batching and caching to scale the method for larger graphs (e.g. ogbn-arxiv). We observe runtime improvements (see Table 2a) and, in some cases, batching with caching is the only method that can run CFGNN (e.g. ogbn-products in Table E1a). Moreover, Figure 6 shows that APS Randomized is only slightly less efficient than CFGNN without the same computational demands. Thus, scaling makes CFGNN more viable for settings with limited resources.
- **Impact of Label Distribution.** We observe that the dataset's label distribution impacts the difference in adaptability between TPS and TPS-Classwise. In Figure 1, we find an increase in the adaptability of TPS when using LC split instead of FS split for ogbn-arxiv compared to Amazon_Computers. The shift in performance is exacerbated when many classes have little representation (highly imbalanced), as seen in Figure 2. The difference in adaptability is also observed in other non-conformity scores and is particularly pronounced for the ogbn-products dataset (see Figure D1 which has several classes with little representation.
- **Incorrect label miscoverages are important.** Conformal prediction is concerned with the true label miscoverage; however, since prediction sets also include incorrect labels, the miscoverage levels of incorrect labels are important to account for when considering a method's efficiency. With Theorem 3.1, we prove that a sufficient condition for when a method is more efficient than another method depends on the incorrect label miscoverage. Theorem 3.1 applies to other domains outside the graph space. We also see in Corollary 3.1.1 that the incorrect label miscoverages determine how much the efficiency improves with large calibration sets. This improvement is pronounced for datasets with many classes.

Guidelines. To conclude this analysis, if computational resources are not a limiting factor, CFGNN with the appropriate loss functions (e.g. APS randomized) has demonstrable success in improving efficiency, while being comparable to other methods in terms of adaptability. When deciding which method is more efficient in general, even for non-graph CP, Theorem 3.1 can be applied to the calibration set to determine this prior to deployment. If adaptability is critical, then methods such as TPS-Classwise and DTPS are both viable and the only methods that provide *a guarantee* while not being as computationally expensive. APS and APS Randomized are good middle-ground methods and NAPS is the only method that provides guarantees for non-exchangeable settings. Lastly, as part of this work, we provide a Python library[3] which can be used to implement these methods.

[3]Code can be found in the supplemental material

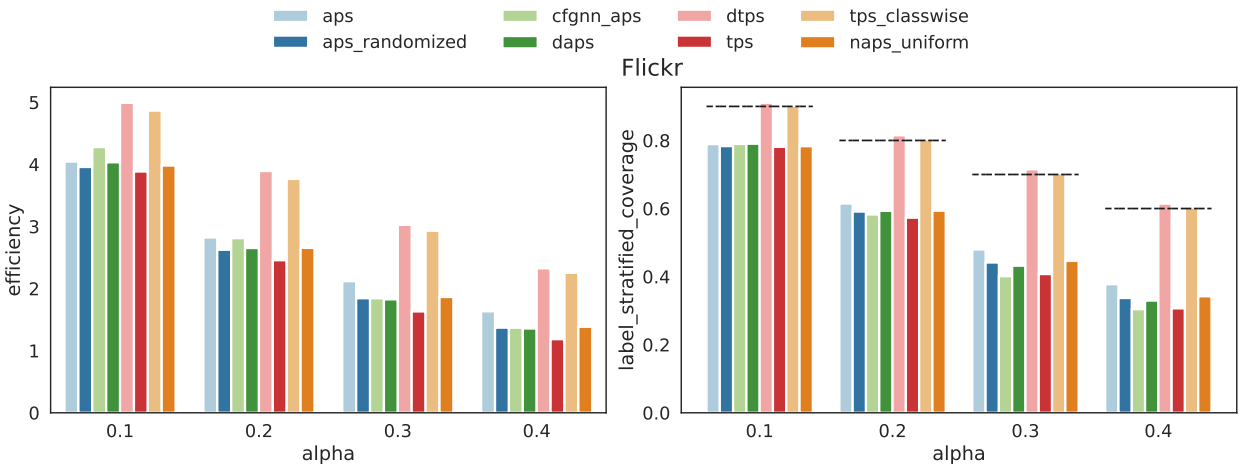


Figure 7: Efficiency (left) and Label Stratified Coverage (right) for all the conformal methods for the Flickr dataset. This uses the FS split style with multiple values for α. The dashed black line indicates the desired label stratified coverage. Other NAPS variants (i.e. exponential, hyperbolic) are discussed in Appendix C.

4 Related Works

4.1 Conformal Prediction for Graphs

In this work, we focus on the prominent methods of graph conformal prediction for the node classification task under a transductive setting. These included standard conformal prediction methods like TPS and APS and graph-specific methods like DAPS and CFGNN. Some works consider other graph-based scenarios and tasks. Within node classification, we can also consider the inductive setting, where nodes/edges arrive in a sequence, thus violating the exchangeability assumption. One such method is Neighborhood Adaptive Prediction Sets (NAPS), which we consider a transductive variation in this work. Beyond node classification on static graphs, there is some work in link prediction (Zhao et al., 2024; Marandon, 2024) as well as with dynamic graphs (Davis et al., 2024). These settings are newly or infrequently studied due to additional assumptions required for meaningful analysis. Thus, we omit these settings and leave them as future work.

4.2 Uncertainty Quantification for Graphs

Beyond conformal prediction, many works propose ways to construct model-agnostic uncertainty estimates for classification tasks (Abdar et al., 2021; Guo et al., 2017; Kull et al., 2019). There are also methods specific to GNNs (Hsu et al., 2022; Wang et al., 2021) that leverage network principles (e.g. homophily). However, unlike conformal prediction, these methods can fail to provide the desired coverage guarantees. For an empirical comparison of popular UQ methods for GNN, please reference Huang et al. (2023).

5 Concluding Remarks

We present a comprehensive benchmarking study of conformal prediction for node classification. We provide novel insights related to design choices that impact efficiency, adaptability, and scalability. Along the way, we offer a new theoretical rationale for the importance of randomization and discuss some novel methodological improvements and directions for future work. One future direction pertains to the space of fairness auditing. Several works have dealt with the auditing fairness of ML models through measuring uncertainty in fairness definitions (Ghosh et al., 2021; Maneriker et al., 2023; Yan & Zhang, 2022), but they rely on the assumption of IID. While conformal prediction works with the notion of *miscoverage*, more relevant notions of error can be considered using the generalized framework of conformal risk control (Angelopoulos et al., 2024).

References

Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Inf. Fusion*, 76(C):243–297, December 2021. ISSN 1566-2535. doi: 10.1016/j.inffus.2021.05.008. URL https://doi.org/10.1016/j.inffus.2021.05.008.

Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

Anastasios N Angelopoulos, Stephen Bates, et al. Conformal prediction: A gentle introduction. *Foundations and Trends® in Machine Learning*, 16(4):494–591, 2023.

Anastasios Nikolas Angelopoulos, Stephen Bates, Michael Jordan, and Jitendra Malik. Uncertainty sets for image classifiers using conformal prediction. In *International Conference on Learning Representations*, 2021.

Anastasios Nikolas Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. In *The Twelfth International Conference on Learning Representations*, 2024.

Anonymous. Removed for blind review. 2025.

Rina Foygel Barber, Emmanuel J Candès, Aaditya Ramdas, and Ryan J Tibshirani. Predictive inference with the jackknife+. *The Annals of Statistics*, 49(1):486–507, 2021.

Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Conformal prediction beyond exchangeability. *The Annals of Statistics*, 51(2):816 – 845, 2023. doi: 10.1214/23-AOS2276. URL https://doi.org/10.1214/23-AOS2276.

Jase Clarkson. Distribution free prediction sets for node classification. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 6268–6278. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/clarkson23a.html.

Ed Davis, Ian Gallagher, Daniel John Lawson, and Patrick Rubin-Delanchy. Valid conformal prediction for dynamic gnns, 2024. URL https://arxiv.org/abs/2405.19230.

Bishwamittra Ghosh, Debabrota Basu, and Kuldeep S Meel. Justicia: A stochastic sat approach to formally verify fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7554–7563, 2021.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pp. 1321–1330. PMLR, 2017.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

Hans Hao-Hsun Hsu, Yuesong Shen, Christian Tomani, and Daniel Cremers. What makes graph neural networks miscalibrated? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 13775–13786. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5975754c7650dfee0682e06e1fec0522-Paper-Conference.pdf.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Kexin Huang, Ying Jin, Emmanuel Candes, and Jure Leskovec. Uncertainty quantification over graph with conformalized graph neural networks. 36:26699–26721, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/54a1495b06c4ee2f07184afb9a37abda-Paper-Conference.pdf.

Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/8ca01ea920679a0fe3728441494041b9-Paper.pdf.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Clemence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs, 2009.

Pranav Maneriker, Codi Burley, and Srinivasan Parthasarathy. Online fairness auditing through iterative refinement. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 1665–1676, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599454. URL https://doi.org/10.1145/3580305.3599454.

Ariane Marandon. Conformal link prediction for false discovery rate control, 2024. URL https://arxiv.org/abs/2306.14693.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

M. Newman. *Networks*. OUP Oxford, 2018. ISBN 9780192527493. URL https://books.google.com/books?id=YdZjDwAAQBAJ.

Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694–706):289–337, February 1933. ISSN 2053-9258. doi: 10.1098/rsta.1933.0009. URL http://dx.doi.org/10.1098/rsta.1933.0009.

Harris Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In *Tools in artificial intelligence*. Citeseer, 2008.

Harris Papadopoulos, Vladimir Vovk, and Alex Gammerman. Qualified predictions for large data sets in the case of pattern recognition. In M. Wani, H. Arabnia, K. Cios, K. Hafeez, and G. Kendall (eds.), *Proceedings of the International Conference on Machine Learning and Applications*, pp. 159–163. CSREA Press, 2002. Proceedings of the International Conference on Machine Learning and Applications, CSREA Press, Las Vegas, NV, pages 159-163, 2002.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks, 2020.

Yaniv Romano, Matteo Sesia, and Emmanuel Candes. Classification with valid and adaptive coverage. *Advances in Neural Information Processing Systems*, 33:3581–3591, 2020.

Mauricio Sadinle, Jing Lei, and Larry Wasserman. Least ambiguous set-valued classifiers with bounded error levels. *Journal of the American Statistical Association*, 114(525):223–234, 2019.

Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

David Stutz, Krishnamurthy Dj Dvijotham, Ali Taylan Cemgil, and Arnaud Doucet. Learning optimal conformal classifiers. In *International Conference on Learning Representations*, 2021.

Vladimir Vovk. Conditional validity of inductive conformal predictors. In *Asian conference on machine learning*, pp. 475–490. PMLR, 2012.

Vladimir Vovk. Cross-conformal predictors. *Annals of Mathematics and Artificial Intelligence*, 74:9–28, 2015.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 02 2020. ISSN 2641-3337. doi: 10.1162/qss_a_00021. URL https://doi.org/10.1162/qss_a_00021.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. Be confident! towards trustworthy graph neural networks via confidence calibration. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 23768–23779. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/c7a9f13a6c0940277d46706c7ca32601-Paper.pdf.

Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph neural networks. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 27–37, 2022.

Tom Yan and Chicheng Zhang. Active fairness auditing. In *International Conference on Machine Learning*, pp. 24929–24962. PMLR, 2022.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Soroush H. Zargarbashi, Simone Antonelli, and Aleksandar Bojchevski. Conformal prediction sets for graph neural networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12292–12318. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/h-zargarbashi23a.html.

Soroush H. Zargarbashi, Mohammad Sadegh Akhondzadeh, and Aleksandar Bojchevski. Robust yet efficient conformal prediction sets. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=MrNq6rbcUi.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method, 2020.

Tianyi Zhao, Jian Kang, and Lu Cheng. Conformalized link prediction on graph neural networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 4490–4499, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3672061. URL https://doi.org/10.1145/3637528.3672061.

A Optimal τ for APS

The most popular baseline in the graph conformal prediction literature is Adaptive Prediction Sets (APS). (Romano et al., 2020) introduce APS by defining an optimal prediction set construction mechanism under oracle probability. Suppose we estimate a prediction function $\hat{f}$ that correctly models the oracle probability $\Pr[Y = y|X_{test} = \boldsymbol{x}] = \pi_y(\boldsymbol{x})$ for each $y \in \mathcal{Y} = \{1, \ldots, K\}$ Let $\pi_{(1)}(\boldsymbol{x}), \ldots, \pi_{(K)}(\boldsymbol{x})$ be the sorted probabilities in descending order. For any $\tau \in [0, 1]$, define the generalized conditional quantile funciton at τ as

$$L(x; \pi, \tau) = \min \left\{ k \in \{1, \ldots, K\}, \sum_{j=1}^{k} \pi_{(j)}(\boldsymbol{x}) \geq \tau \right\} \tag{8}$$

The corresponding prediction set, $C_\alpha^{\text{or}}(\boldsymbol{x})$, is constructed as

$$C_\alpha^{\text{or}+}(\boldsymbol{x}) = \{y \in \mathcal{Y} : \pi_y(\boldsymbol{x}) \geq \pi_{(L(\boldsymbol{x};\pi,1-\alpha))}(\boldsymbol{x})\}$$

where or indicates the usage of the oracle probability. Further, they define tighter prediction sets in a randomized fashion using an additional uniform random variable $u \sim \text{Uniform}(0, 1)$ as a parameter to construct a generalized inverse. This idea draws upon the idea of uniformly most powerful tests in the Neyman-Pearson lemma for level-α sets (Neyman & Pearson, 1933). Define

$$S(\boldsymbol{x}, u; \pi, \tau) = \begin{cases} \{y \in \mathcal{Y} : \pi_y(\boldsymbol{x}) > \pi_{(L(\boldsymbol{x};\pi,\tau))}(\boldsymbol{x})\} & u < V(\boldsymbol{x}; \pi, \tau) \\ \{y \in \mathcal{Y} : \pi_y(\boldsymbol{x}) \geq \pi_{(L(\boldsymbol{x};\pi,\tau))}(\boldsymbol{x})\} & \text{otherwise} \end{cases} \tag{9}$$

i.e., the class at the $L(\boldsymbol{x}; \pi, \tau)$ rank is included in the prediction set with probability $1 - V(\boldsymbol{x}; \pi, \tau)$, where

$$V(\boldsymbol{x}; \pi, \tau) = \frac{1}{\pi_{(L(\boldsymbol{x};\pi,\tau))}(x)} \left\{ \left[\sum_{j=1}^{L(\boldsymbol{x};\pi,\tau)} \pi_{(j)}(x) \right] - \tau \right\}$$

The corresponding randomized prediction sets are $C_\alpha^{\text{or}}(\boldsymbol{x}) = S(\boldsymbol{x}, U; \pi, 1-\alpha)$, $U \sim U(0, 1)$ Note that in general, the coverage guarantees provided in conformal prediction hold only in expectation over the randomness in $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n + 1$. The randomized prediction sets continue to provide the guarantee with additional randomness over u_i. To make this work for a non-oracle probability $\hat{\pi}(\boldsymbol{x})$, they define a non-conformity score A

$$A(\boldsymbol{x}, y, u; \hat{\pi}) = \min\{\tau \in [0, 1] : y \in S(\boldsymbol{x}, u; \hat{\pi}, \tau)\} \tag{10}$$

Assume that $\hat{\pi}$ are all distinct - for ease of defining rank. Suppose the rank of the true class amongst the sorted $\hat{\pi}$ be r_y, i.e., $\sum_{i=1}^{K} \mathbf{1}[\hat{\pi}_i(\boldsymbol{x}) \geq \hat{\pi}_y] = r_y$ Solving for τ as a function of $\hat{\pi}$ (see Appendix A, for proof),

$$A(\boldsymbol{x}, y, u; \hat{\pi}) = \left[\sum_{i=1}^{r_y} \hat{\pi}_{(i)}(\boldsymbol{x}) \right] - u\hat{\pi}_y \tag{11}$$

Instead, if a deterministic set is used to define the conformal score instead (i.e., the randomized set construction is not carried out), then we could just add the probabilities until the true class is included:

$$\tilde{A}(\boldsymbol{x}, y; \hat{\pi}) = \left[\sum_{i=1}^{r_y} \hat{\pi}_{(i)}(\boldsymbol{x}) \right] \tag{12}$$

This version of APS still provides the same conditional coverage guarantees and has a simpler exposition as the prediction sets are constructed by greedily including the classes until the true label is included. Thus, this version is provided as the implementation in the popular monographs on conformal prediction by Angelopoulos

& Bates (2021) and Angelopoulos et al. (2023). However, the lack of randomization may sacrifice efficiency. This modification of the score function affects both the quantile threshold computation during the calibration phase and the prediction set during the test phase. We will now show the conditions that impact the efficiency more formally.

For simplicity, assume that the probabilities are distinct.

From the definition of A equation 10

$$A(\boldsymbol{x}, y, u; \hat{\pi}) = \min\{\tau \in [0,1] : y \in S(\boldsymbol{x}, u; \hat{\pi}, \tau)\}$$

Define

$$\Sigma_{\hat{\pi}}(\boldsymbol{x}, m) = \sum_{i=1}^{m} \hat{\pi}_{(i)}(\boldsymbol{x})$$

From the definition of $S(\boldsymbol{x}, u; \hat{\pi}, \tau)$ from equation 9, conisder the following cases:

Case 1: $\tau = \Sigma_{\hat{\pi}}(\boldsymbol{x}, r_y)$, then $L(x; \hat{\pi}, \tau) = y$ and thus, $V(\boldsymbol{x}; \pi, \tau) = 0$. Thus $\Pr[u > V(\boldsymbol{x}; \pi, \tau)] = 1$ and hence, $P[y \in S(\boldsymbol{x}, u; \hat{\pi}, \tau)] = 1$.

Case 2: $\tau = \Sigma_{\hat{\pi}}(\boldsymbol{x}, r_y - 1)$, then $y \notin S(\boldsymbol{x}, u, \hat{\pi}, \tau)$ in either case, since only classes with $\hat{\pi}_i(\boldsymbol{x}) > \hat{\pi}_y(\boldsymbol{x})$ could be included.

Case 3: $\tau = \Sigma_{\hat{\pi}}(\boldsymbol{x}, r_y) - \varepsilon\hat{\pi}_y$. Then we have $L(x; \hat{\pi}, \tau) = y$ again, and

$$\begin{aligned} V(\boldsymbol{x}; \pi, \tau) &= \frac{1}{\hat{\pi}_y(\boldsymbol{x})} \left\{ \left[\sum_{j=1}^{r_y} \hat{\pi}_{(j)}(\boldsymbol{x}) \right] - \tau \right\} \\ &= \frac{1}{\hat{\pi}_y(\boldsymbol{x})} \left\{ \left[\sum_{j=1}^{r_y} \hat{\pi}_{(j)}(\boldsymbol{x}) \right] - (\Sigma_{\hat{\pi}}(\boldsymbol{x}, r_y) - \varepsilon\hat{\pi}_y) \right\} \\ &= \varepsilon \end{aligned}$$

For y to be included in $S(\boldsymbol{x}, u; \hat{\pi}, \tau)$, we would require that $u \geq V(\boldsymbol{x}; \pi, \tau)$, i.e., $u \geq \varepsilon$. We want the minimal τ (or the maximal ε). Thus, $\tau = \Sigma_{\hat{\pi}}(\boldsymbol{x}, r_y) - u\hat{\pi}_y$ is the required solution.

A.1 Non-randomized set

The inclusion criterion for the score given the threshold τ is $\tilde{A}(\boldsymbol{x}, y; \hat{pi}) \leq \tau$

To include the correct label y_i while minimizing the chosen threshold τ, we would require $\tau = \sum_{j=1}^{r_{y_i}} \hat{\pi}_{(j)}(\boldsymbol{x})$

B Proofs

B.1 Proof of Lemma 3.1

To prove Lemma 3.1, we use the following lemma:

Lemma B.1. *For, exchangeable random variables,* $Z_1, Z_2, \ldots, Z_n$, $\Pr[Z_n \leq \lambda] = \frac{\sum_{i=1}^{n} \mathbf{1}[Z_i \leq \lambda]}{n}$.

Proof. We define the *quantile function*, $Q(\alpha)$, and the *right quantile function*, $R(\alpha)$ such that

$$Q(\alpha) \leq z \iff \alpha \leq F(z)$$

$$z \leq R(\alpha) \iff F^{-}(z) \leq \alpha$$

where $F(z) := \Pr(Z \leq z)$, $F^{-}(z) := \Pr(Z < z)$, $\alpha \in [0, 1]$. For an empirical quantile function $\hat{Q}_n$, we use the empirical CDF $\hat{F}_n := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{Z_i \leq z}$, similarly we define $\hat{R}_n$ and $\hat{F}_n^{-} := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{Z_i < z}$. Observe that $\hat{Q}_n(\alpha) = Z_{(\lceil \alpha n \rceil)}$ and $\hat{R}_n(\alpha) = Z_{(\lfloor \alpha n \rfloor + 1)}$.

Let $k = \sum_{i=1}^{n} \mathbf{1}[Z_i \leq \lambda]$ and $\alpha = \frac{k}{n}$. Then $\hat{Q}_n(\alpha) = Z_{(k)}$ and $\hat{R}_n(\alpha) = Z_{(k+1)}$, resulting in $\hat{Q}_n(\alpha) < \lambda < \hat{R}_n(\alpha)$.

By exchangeability, we have that $\Pr[Z_i \leq \lambda]$ is equal for all $1 \leq i \leq n$. Thus,

$$\mathbb{E}[\hat{F}_n(\lambda)] = \frac{1}{n}\sum_{i=1}^{n} \Pr[Z_i \leq \lambda] = \Pr[Z_n \leq \lambda]$$

As defined, $\hat{F}_n(\hat{Q}_n(\alpha)) \geq \alpha$ and $\hat{F}_n^{-}(\hat{R}_n(\alpha)) \leq \alpha$, since $\hat{Q}_n(\alpha) < \lambda < \hat{R}_n(\alpha)$. Observe that $\hat{F}_n(\hat{Q}_n(\alpha)) = \hat{F}_n^{-}(\hat{R}_n(\alpha))$, since $\hat{Q}_n(\alpha) = Z_{(k)}$ and $\hat{R}_n(\alpha) = Z_{(k+1)}$. Additionally, $\hat{F}_n(\hat{Q}_n(\alpha)) = \hat{F}_n(\lambda)$ since there are no additional elements between $Z_{(k)}$ and λ. Then $\alpha \leq \hat{F}_n(\lambda) \leq \alpha$. Thus by applying the expectation, and the squeeze theorem we get

$$\Pr[Z_n \leq \lambda] = \frac{\sum_{i=1}^{n} \mathbf{1}[Z_i \leq z]}{n}.$$

□

Now using Lemma B.1, we will prove Lemma 3.1.

Lemma 3.1 (Anonymous (2025))**.** *For* $\lambda \in [0, 1]$ *and* $n = |\mathcal{D}_{\text{calib}}|$,

$$\frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) > \lambda]}{n+1} \leq \Pr[y_{n+1} \notin \mathcal{C}_\lambda(\boldsymbol{x}_{n+1})] \leq \frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) > \lambda] + 1}{n+1} \tag{6}$$

Proof. Using Lemma B.1 on the sequence of exchangeable random variables, $\{s(\boldsymbol{x}_i, y_i)\}_{i=1}^{n+1}$, we get that $\Pr[s(\boldsymbol{x}_{n+1}, y_{n+1}) \leq \lambda] = \frac{\sum_{i=1}^{n+1} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda]}{n+1}$. Since $s(\boldsymbol{x}_{n+1}, y_{n+1})$ is unknown, we bound $\frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda]}{n+1} \leq \frac{\sum_{i=1}^{n+1} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda]}{n+1} \leq \frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda] + 1}{n+1}$. Thus,

$$\frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda]}{n+1} \leq \Pr[s(\boldsymbol{x}_{n+1}, y_{n+1}) \leq \lambda] \leq \frac{\sum_{i=1}^{n} \mathbf{1}[s(\boldsymbol{x}_i, y_i) \leq \lambda] + 1}{n+1} \tag{13}$$

Since $\Pr[s(\boldsymbol{x}_{n+1}, y_{n+1}) \leq \lambda] = \Pr[y_{n+1} \in \mathcal{C}_\lambda(x)]$, considering the complement event gives 6. □

B.2 Proof of Theorem 3.1

For self-containment, we will restate some of the notation, used in proving Theorem 3.1. Let $A(\boldsymbol{x}, y)$ be any non-conformity score function and

$$\hat{q}_A = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha) \rceil}{n}; \{A(\boldsymbol{x}_i, y_i)\}_{i=1}^n\right)$$

Define $A_i(y) := A(\boldsymbol{x}_i, y)$ where $y \in \mathcal{Y}$ and $C_A^i = C_A(\boldsymbol{x}_i)$ for brevity. Let $\mathcal{Y}_i' = \mathcal{Y} \setminus \{y_i\}$ be the set of incorrect labels and $y_i' \in \mathcal{Y}_i'$ be any incorrect class label for each $\boldsymbol{x}_i$. We can consider the exchangeable sequence $\{A(\boldsymbol{x}_i, y_i^R)\}_{i=1}^{n+1}$, where y_i^R denotes a randomly sampled from $\mathcal{Y}_i'$, and define $\alpha_c^A \in [0, 1]$ such that:

$$\hat{q}_A = \text{Quantile}\left(\frac{\lceil (n+1)(1-\alpha_c^A) \rceil}{n};\ \{A(\boldsymbol{x}_i, y_i^R)\}_{i=1}^n\right). \tag{14}$$

To prove Theorem 3.1, we all use the following lemma:

Lemma B.2. *The random variable $y_i^R \sim U(\mathcal{Y}_i')$.*

Proof. To prove the lemma, we will show that the PMF that $y'_{i,(u_i)}$ follows is identical to $U(\mathcal{Y}_i')$. Let y_i' be an arbitrary label in $\mathcal{Y}_i'$. First observe for $y_i^R \sim U(\mathcal{Y}_i')$ that $\Pr[y_i^R = y_i'] = \frac{1}{K-1}$ because it is uniformly sampled. Now consider via the law of total probability,

$$\begin{aligned}
\Pr[y'_{i,(u_i)} = y_i'] &= \sum_{l=1}^{K-1} \Pr[y'_{i,(u_i)} = y_i' | u_i = l] \cdot \Pr[u_i = l] \\
&= \sum_{l=1}^{K-1} \Pr[y'_{i,(l)} = y_i'] \cdot \Pr[u_i = l] \\
&= \sum_{l=1}^{K-1} \Pr[y'_{i,(l)} = y_i'] \frac{1}{K-1} \\
&= \frac{1}{K-1} \underbrace{\sum_{l=1}^{K-1} \Pr[y'_{i,(l)} = y_i']}_{\text{Sum of a PMF}} \\
&= \frac{1}{K-1} \cdot 1 = \frac{1}{K-1}
\end{aligned} \tag{15}$$

Thus, the PMFs are identical and $y_i^R \sim U(\mathcal{Y}_i')$. □

Now, we will prove Theorem 3.1.

Theorem 3.1. *If $\alpha_c^A - \alpha_c^{\tilde{A}} \geq \frac{2}{(n+1)}$ then score function A produces a more efficient prediction set than $\tilde{A}$. Formally, $\mathbb{E}\left[|C_{\tilde{A}}(\boldsymbol{x}_{n+1})| - |C_A(\boldsymbol{x}_{n+1})|\right] \geq 0$*

Proof. First consider, $\mathbb{E}\big[\big|C_A^{n+1}\big|\big]$,

$$\begin{aligned}
\mathbb{E}\big[\big|C_A^{n+1}\big|\big] &= \mathbb{E}\left[\sum_{y\in\mathcal{Y}} \mathbf{1}[y \in C_A^{n+1}]\right] \\
&= \mathbb{E}\left[\mathbf{1}[y_{n+1} \in C_A^{n+1}]\right] + \mathbb{E}\left[\sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \mathbf{1}[y'_{n+1} \in C_A^{n+1}]\right] \\
&= \mathbb{E}\left[\mathbf{1}[y_{n+1} \in C_A^{n+1}]\right] + \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \mathbb{E}\left[\mathbf{1}[y'_{n+1} \in C_A^{n+1}]\right] \\
&= \Pr[y_{n+1} \in C_A^{n+1}] + \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[y'_{n+1} \in C_A^{n+1}]
\end{aligned} \tag{16}$$

Now using Lemma B.2 we have that $y_i^R \sim U(\mathcal{Y}'_i)$. Then, using the law of total probability and the fact that a label y is in C_A^{n+1} iff $A(\boldsymbol{x}_{n+1}, y) \leq \hat{q}_A$,

$$\begin{aligned}
\Pr[A(\boldsymbol{x}_{n+1}, y_{n+1}^R) \leq \hat{q}_A] &= \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[A(\boldsymbol{x}_{n+1}, y_{n+1}^R) \leq \hat{q}_A | y_{n+1}^R = y'_{n+1}] \cdot \Pr[y_{n+1}^R = y'_{n+1}] \\
&= \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[A(\boldsymbol{x}_{n+1}, y'_{n+1}) \leq \hat{q}_A] \cdot \Pr[y_{n+1}^R = y'_{n+1}] \\
&= \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[y'_{n+1} \in C_A^{n+1}] \cdot \Pr[y_{n+1}^R = y'_{n+1}] \\
&= \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[y'_{n+1} \in C_A^{n+1}] \cdot \frac{1}{K-1} \\
\iff (K-1)\Pr[y_{n+1}^R \in C_A^{n+1}] &= \sum_{y'_{n+1}\in\mathcal{Y}'_{n+1}} \Pr[y'_{n+1} \in C_A^{n+1}]
\end{aligned} \tag{17}$$

Then substituting Equation 17 into Equation 16 we arrive at,

$$\mathbb{E}\big[\big|C_A^{n+1}\big|\big] = \Pr[y_{n+1} \in C_A^{n+1}] + (K-1)\Pr[y_{n+1}^R \in C_A^{n+1}] \tag{18}$$

Then we can compute an upper bound and lower bound for $\mathbb{E}\big[\big|C_A^{n+1}\big|\big]$,

$$\begin{aligned}
\mathbb{E}\big[\big|C_A^{n+1}\big|\big] &\leq 1 - \alpha + \frac{1}{n+1} + (K-1)\left(1 - \alpha_c^A + \frac{1}{n+1}\right) \\
&= 1 - \alpha + (K-1)\left(1 - \alpha_c^A\right) + \frac{K}{n+1}
\end{aligned}$$

and

$$\mathbb{E}\big[\big|C_A^{n+1}\big|\big] \geq 1 - \alpha + (K-1)\left(1 - \alpha_c^A\right) \tag{19}$$

similar bounds can be derived for $\mathbb{E}\left[\left|C_{\tilde{A}}^{n+1}\right|\right]$. Thus,

$$\begin{aligned}
\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right] &\geq \underbrace{\left(1-\alpha+(K-1)\left(1-\alpha_c^{\tilde{A}}\right)\right)}_{\text{lower bound for } \mathbb{E}\left[\left|C_{\tilde{A}}^{n+1}\right|\right]} - \underbrace{\left(1-\alpha+(K-1)\left(1-\alpha_c^{A}\right)+\frac{K}{n+1}\right)}_{\text{upper bound for } \mathbb{E}\left[\left|C_{A}^{n+1}\right|\right]} \\
&= (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}}\right) - \frac{K}{n+1} \\
&= (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}} - \frac{K}{(K-1)n+1}\right) \\
&\geq (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}} - \frac{2}{n+1}\right) \geq 0
\end{aligned} \tag{20}$$

with the last inequality holding because $\alpha_c^A - \alpha_c^{\tilde{A}} \geq \frac{2}{n+1}$. This completes the proof for the general case. □

B.3 Proof of Corollary 3.1.1

Corollary 3.1.1. *As $n \to \infty$, $\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right] = (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}}\right)$*

Proof. Observe using Equation 20, $\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right] \geq (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}} - \frac{2}{n+1}\right)$. Similarly by swapping A and $\tilde{A}$:

$$\begin{aligned}
&\mathbb{E}\left[|C_{A}^{n+1}| - |C_{\tilde{A}}^{n+1}|\right] \geq (K-1)\left(\alpha_c^{\tilde{A}} - \alpha_c^{A} - \frac{2}{n+1}\right) \\
\iff &\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right] \leq (K-1)\left(\alpha_c^{A} - \alpha_c^{\tilde{A}} + \frac{2}{n+1}\right)
\end{aligned}$$

Then observe as $n \to \infty$ both the lower and upper bound of $\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right]$ converge to $(K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}}\right)$. Thus $\mathbb{E}\left[|C_{\tilde{A}}^{n+1}| - |C_{A}^{n+1}|\right] = (K-1)\left(\alpha_c^A - \alpha_c^{\tilde{A}}\right)$. □

C Method Details and Innovations

C.1 Notes on Transductive NAPS

Neighborhood Adaptive Prediction Sets (NAPS) constructs prediction sets under relaxed exchangeability (or non-exchangeability) assumptions (Barber et al., 2023) and was initially implemented for the inductive setting (Clarkson, 2023). However, NAPS can also be used in the transductive setting (Zargarbashi et al., 2023). To compute scores for $\mathcal{D}_{\text{calib}}$ nodes, NAPS uses APS. Using these scores, Equation 21 is used to compute a *weighted quantile* for the score threshold to be used when constructing the prediction sets. The weighted quantile is defined by placing a point mass, δ_{s_i}, for each calibration point's score, s_i, as well as a point mass at $_\infty$ to represent the test point, v_{n+1}'s, score. The point mass at $\delta_{+\infty}$ is needed because the score for v_{n+1} is unknown, and potentially unbounded, due to non-exchangeability.

$$\hat{q}^{\text{NAPS}}_{n+1} = \text{Quantile}\left(1-\alpha, \left[\sum_{i \in \mathcal{D}_{\text{calib}}} \tilde{w}_i \cdot \delta_{s_i}\right] + \tilde{w}_{n+1} \cdot \delta_{+\infty}\right) \tag{21}$$

For NAPS to produce viable prediction sets, the weights, $w_i \in [0,1]$, for the calibration nodes must be chosen in a data-independent fashion, i.e., they cannot leverage the associated node features (Barber et al., 2023). NAPS leverages the graph structure to assign these weights, assigning non-zero weights to nodes within a k-hop neighborhood, $\mathcal{N}^k_{n+1}$, of a test node v_{n+1}. For nodes that are in $\mathcal{N}^k_{n+1}$, let d_i be the distance from the node to v_{n+1} to $v_i \in \mathcal{V}_{\text{calib}}$. The three implemented weight functions are *uniform*: $w_u(d_i) = 1$, *hyperbolic*: $w_h(d_i) = \frac{1}{d_i}$, and *exponential*: $w_e(d_i) = 2^{-d_i}$. Nodes that are not in the $\mathcal{N}^k_{n+1}$ have zero weight. The weights are then normalized, $\tilde{w}_i$, such that $\sum_{i \in \mathcal{D}_{\text{calib}}} \tilde{w}_i + \tilde{w}_{n+1} = 1$ (Barber et al., 2023).

NAPS Implementation NAPS is computationally expensive in terms of time and memory since the k-hop intersection is computed for each test node. To allow for scalability, our implementation of NAPS, shown in Algorithms 1 and 2, uses batching to ensure sufficient memory is available and uses sparse-tensor multiplication to reduce memory and time costs.

To ensure scalability for large graphs, all the computations until the quantile computation step were done via sparse tensors. Algorithm 2 illustrates how the distance to each calibration node in the k-hop neighborhood can be computed via sparse tensor primitives.

Algorithm 1 NAPS Quantile Implementation

procedure NAPS_QUANTILE($w, k, \mathcal{D}_{\text{calib}}, \mathcal{D}_{\text{test}}, \mathcal{D}, \mathcal{S}_{\text{calib}}, b, \alpha$)
  $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_b\} \leftarrow$ SPLIT($\mathcal{D}_{\text{test}}, b$) ▷ Split test nodes into b batches
  $q \leftarrow$ ZEROS($\mathcal{D}_{\text{test}}, 1$) ▷ $q \in \mathbb{R}^{|\mathcal{D}_{\text{test}}| \times 1}$
  for $\mathcal{B}_n \in \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_b\}$ **do**
    k_hop $\leftarrow$ SPARSE_K_HOP($k, \mathcal{B}_n, \mathcal{D}_{\text{calib}}, \mathcal{D}$) ▷ k_hop $\in \mathbb{R}^{|\mathcal{B}_n| \times |\mathcal{D}_{\text{calib}}|}$
    weights $\leftarrow$ COMPUTE_WEIGHTS(w, k_hop) ▷ weights $\in \mathbb{R}^{|\mathcal{B}_n| \times |\mathcal{D}_{\text{calib}}|}$
    $q[\mathcal{B}_n] \leftarrow$ COMPUTE_QUANTILE($1-\alpha$, weights, $\mathcal{S}_{\text{calib}}$)
  end for
  return q ▷ Return the quantiles for each test node
end procedure

Parameter Analysis: Apart from the particular weighting function, the main parameter in the NAPS algorithm is the number of hops to consider, k. Figure C1 shows the trend between efficiency and coverage as we increase k from 1 to D, where D is a lower bound on the diameter of the largest strongly connected component of the dataset, computed using the NetworkX Hagberg et al. (2008). For each dataset, we observe a value of k after which the efficiency does not improve, while still achieving the desired coverage. This behavior can suggest a heuristic akin to the 'elbow method' used in clustering analysis to determine the number of clusters for choosing the value of k in NAPS.

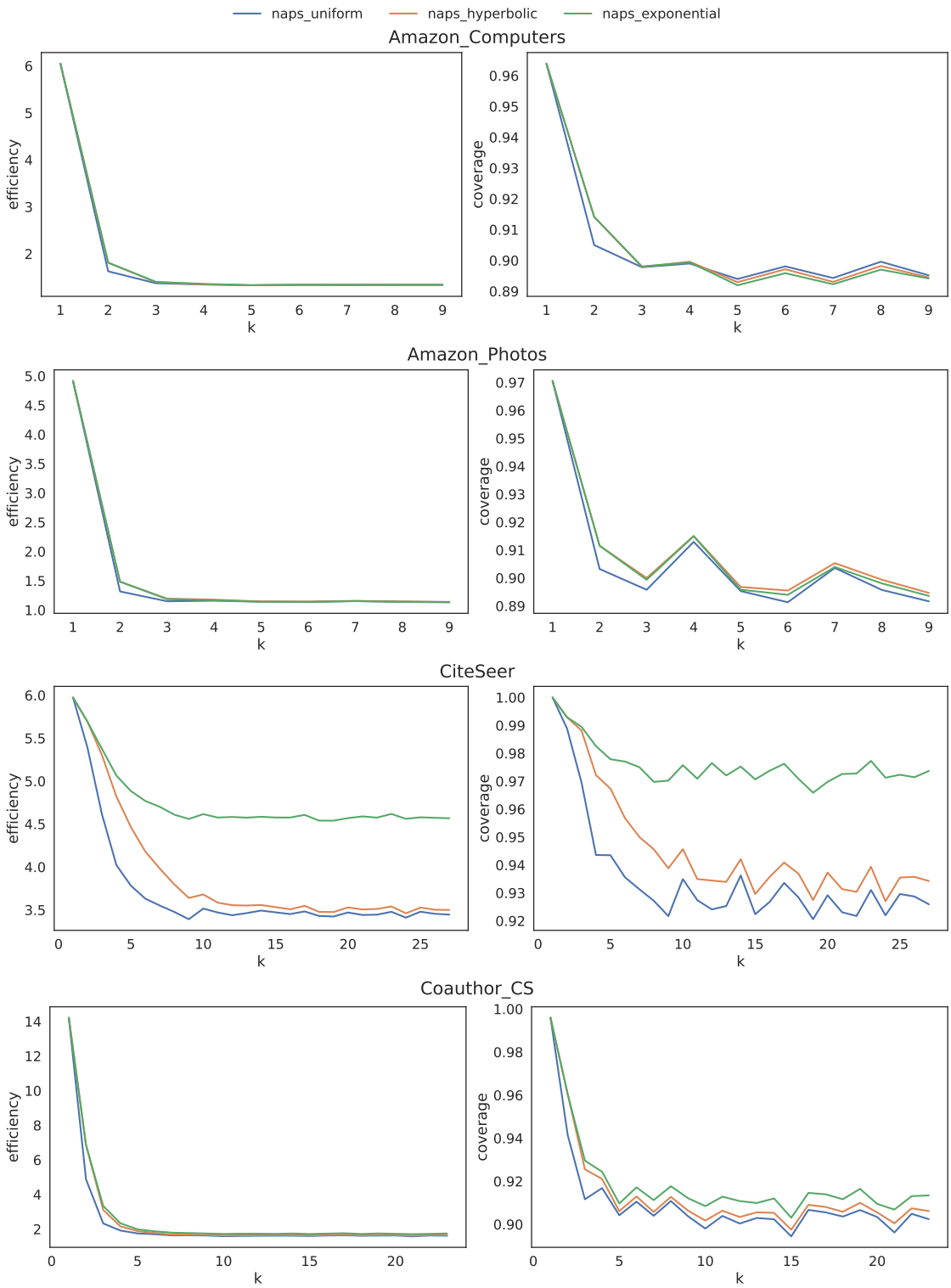


Figure C1: Plotting the Efficiency and Coverage when using NAPS for k from 1 to D. The above results are with FS split and $\alpha = 0.1$.

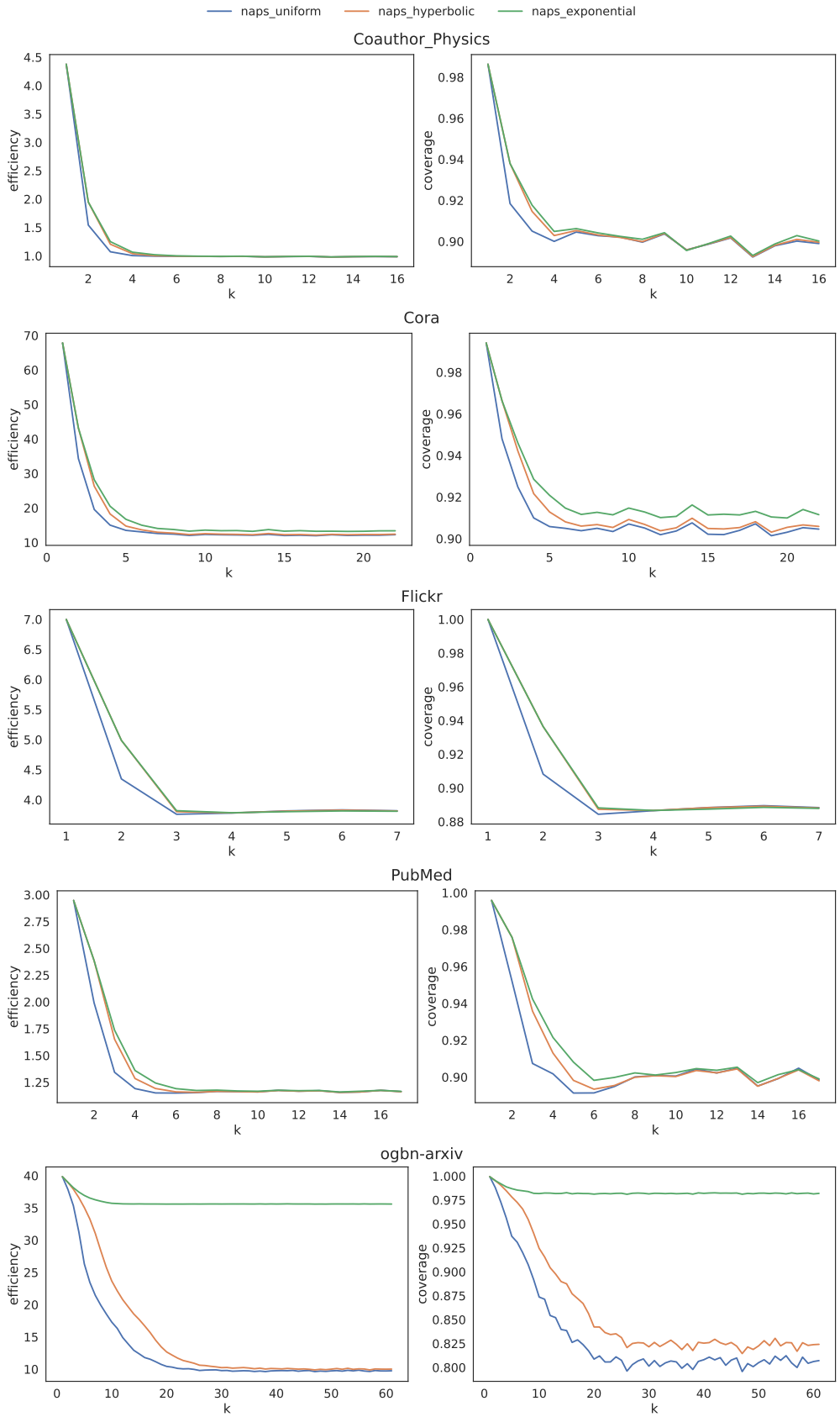


Figure C1: Plotting the Efficiency and Coverage when using NAPS for k from 1 to D. The above results are with FS split and $\alpha = 0.1$ (cont.). The ogbn-products dataset is omitted due to size and lack of data points.

Algorithm 2 Sparse K Hop Neighborhood Implementation

procedure SPARSE_K_HOP($k, \mathcal{B}, \mathcal{D}_{\text{calib}}, \mathcal{D}$)
  A $\leftarrow$ GET_ADJACENCY($\mathcal{D}$) ▷ Adjacency of $\mathcal{D}$, A $\in \mathbb{R}^{|\mathcal{D}|\times|\mathcal{D}|}$
  path_n $\leftarrow$ A[$\mathcal{B}$, :] ▷ path_n $\in \mathbb{R}^{|\mathcal{B}|\times|\mathcal{D}|}$
  k_hop $\leftarrow$ path_n[:, $\mathcal{D}_{\text{calib}}$] ▷ k_hop $\in \mathbb{R}^{|\mathcal{B}|\times|\mathcal{D}_{\text{calib}}|}$
  for $n \in \{2, 3, \ldots, k\}$ **do**
    path_n $\leftarrow$ (path_n)A
    neg_if_n $\leftarrow$ k_hop $-$ SGN(path_n[:, $\mathcal{D}_{\text{calib}}$]) ▷ negative value $\implies$ n hops away
    in_n_hop $\leftarrow$ (neg_if_n < 0) $\times\, n$ ▷ Nodes that are a min distance of n
    k_hop $\leftarrow$ k_hop + in_n_hop
  end for
  return k_hop ▷ $\forall_{i,j}$ **If** dist$(i, j) \leq k$ **then** k_hop$[i, j]$ = dist(i, j), **else** k_hop$[i, j] = 0$
end procedure

C.2 Diffusion Adaptive Prediction Sets (DAPS)

The Diffusion Adaptive Prediction Sets (DAPS) approach for conformal node classification on graphs was introduced by (Zargarbashi et al., 2023). The intuition behind DAPS follows the prevalence of homophily graphs, which suggests that non-conformity scores for two connected nodes should be related. DAPS uses a diffusion step to capture this relationship and uses the non-conformity scores modified by diffusion to generate the prediction sets. Formally, suppose $s(v, y)$ is a point wise non-conformity score for a node v and label y (e.g., TPS or APS)

$$\hat{s}(v, y) = (1 - \lambda)s(v, y) + \frac{\lambda}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} s(u, y)$$

where $\mathcal{N}_v$ is the 1-hop neighborhood of v and $\lambda \in [0, 1]$ is a hyperparameter controlling the diffusion.

Zargarbashi et al. (2023) use the APS score as the point-wise score in the diffusion process as it is adaptive and uniformly distributed in $[0, 1]$ under oracle probability. However, as noted earlier, using classwise thresholds provides a mechanism to produce adaptive scores from TPS. Thus, we create DTPS, a variation of DAPS using TPS scores as the point-wise scores in the diffusion process.

We compare our proposed method of using diffusion on top of TPS-Classwise (DTPS) against DAPS, which was proposed by Zargarbashi et al. (2023). From Figure C2 (left), we see that DTPS can be competitive with DAPS in efficiency while providing better label-stratified coverage. However, for some of the larger datasets (Cora, Flickr, ogbn-arxiv, ogbn-products), DTPS suffers from poorer efficiency compared to DAPS. This can be partially explained by the worse performance of the pre-diffusion TPS-classwise (Figure E2) which is forced to sacrifice efficiency on these datasets to achieve label-stratified coverage (Figure E1). However, when we control the number of samples per class with LC splits (Figure C2 right), we see that DTPS label stratified coverage deteriorates significantly compared to DAPS. Based on these results, we can conclude that DTPS is not a universally better method than DAPS, and its performance is sensitive to the calibration set size and the number of classes. It may be a viable candidate over DAPS in scenarios when there is a sufficiently large calibration set, which is when TPS-classwise has competitive efficiency to TPS.

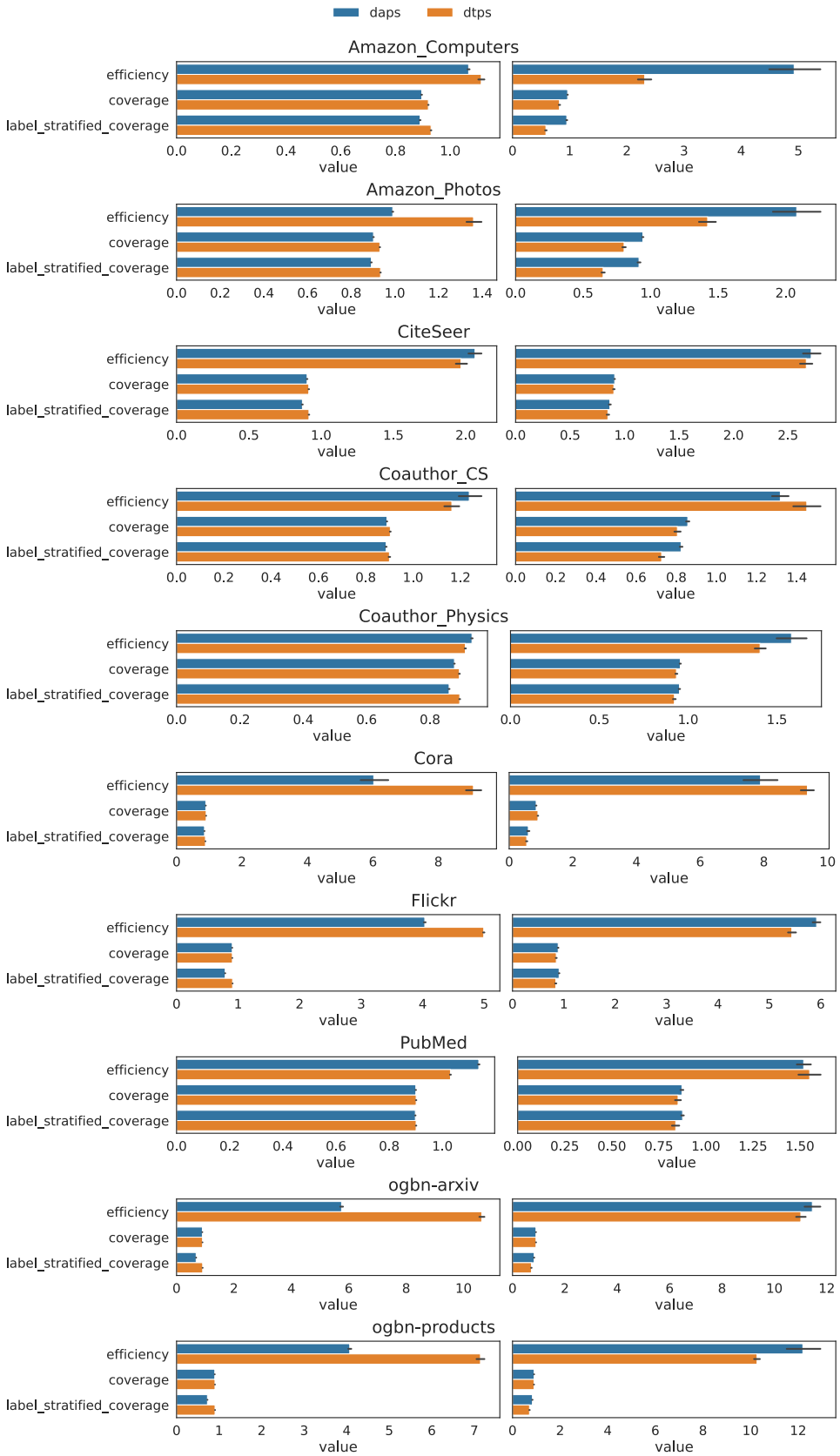


Figure C2: Bar charts denoting different metrics associated with DAPS and DTPS across all datasets for FS split (left) and LC split (right) at $\alpha = 0.1$.

D Datasets and Hyperparameter Tuning Details

D.1 Datasets

We selected datasets of varying sizes and origins to evaluate the performance of the graph conformal prediction methods. The first category of datasets is citation datasets, where the nodes are publications and the edges denote citation relationships. Nodes have features that are bag-of-words representations of the publication. The task is to predict the category of each publication. The citation networks we use are **CiteSeer** (Yang et al., 2016), CoraFull (Shchur et al., 2018), an extended version of the common Cora (Yang et al., 2016) citation network dataset, and **Pubmed** (Yang et al., 2016). The second category comes from the Amazon Co-Purchase graph (McAuley et al., 2015), where nodes represent goods, edges represent goods frequently bought together, and node features are bag-of-words representations of product reviews. The task is to predict the category of a good. We use the **Amazon_Photos** and **Amazon_Computers** datasets. The last category is co-authorship networks extracted from the Microsoft Academic Graph (Wang et al., 2020) used for KDD Cup'16. The nodes are authors, edges represent coauthorship, and node features represent paper keywords of the author's publications. The task is to predict the author's most active field of study. We use **Coauthor_CS** and **Coauthor_Physics** which both come from the Microsoft Academic Graph (Wang et al., 2020). Other datasets that were use include **Flickr** (Zeng et al., 2020), **ogbn-arxiv**, and **ogbn-products** Hu et al. (2020). These last three datasets have predefined splits for train/validation/test, shown in Table D2, which we used when constructing our train/validation/calibration/test splits for the different split styles. We used the version given by the Deep Graph Library (Wang et al., 2019) for all the datasets. DGL uses an Apache 2.0 license, and OGB uses an MIT license.

Table D1 presents summary statistics for each dataset. These include the average local clustering coefficient (Avg CC), global clustering coefficient (Global CC) (Newman, 2018), an approximate lower bound on the diameter (D) given by Magnien et al. (2009), node homophily ratio ($\hat{H}$) (Pei et al., 2020), and expected node homophily ratio (H_{rand}). Figure D1 shows the label distribution for each dataset.

Table D1: Summary statistics for all datasets evaluated.

Dataset	Nodes	Edges	Classes	Features	Avg CC	Global CC	D	$\hat{H}$	H_{rand}
CiteSeer	3,327	9,228	6	3,703	0.141	0.130	28	0.722	0.178
Amazon_Photos	7,650	238,163	8	745	0.404	0.177	10	0.836	0.165
Amazon_Computers	13,752	491,722	10	767	0.344	0.108	10	0.785	0.208
Cora	19,793	126,842	70	8,710	0.261	0.131	23	0.586	0.022
PubMed	19,717	88,651	3	500	0.060	0.054	17	0.792	0.357
Coauthor_CS	18,333	163,788	15	6,805	0.343	0.183	24	0.832	0.112
Coauthor_Physics	34,493	495,924	5	8,415	0.378	0.187	17	0.915	0.321
Flickr	89,250	899,756	7	500	0.033	0.004	8	0.322	0.267
ogbn-arxiv	169,343	1,166,243	40	128	0.118	0.115	62	0.567	0.077
ogbn-products	2,449,029	61,859,140	47	100	0.411	0.130	27	0.817	0.106

Table D2: Predefined splits from original source noted.

Dataset	# Train	# Valid	# Test
Flickr	44,625	22,312	22,313
ogbn-arxiv	90,941	29,799	48,603
ogbn-products	196,615	39,323	2,213,091

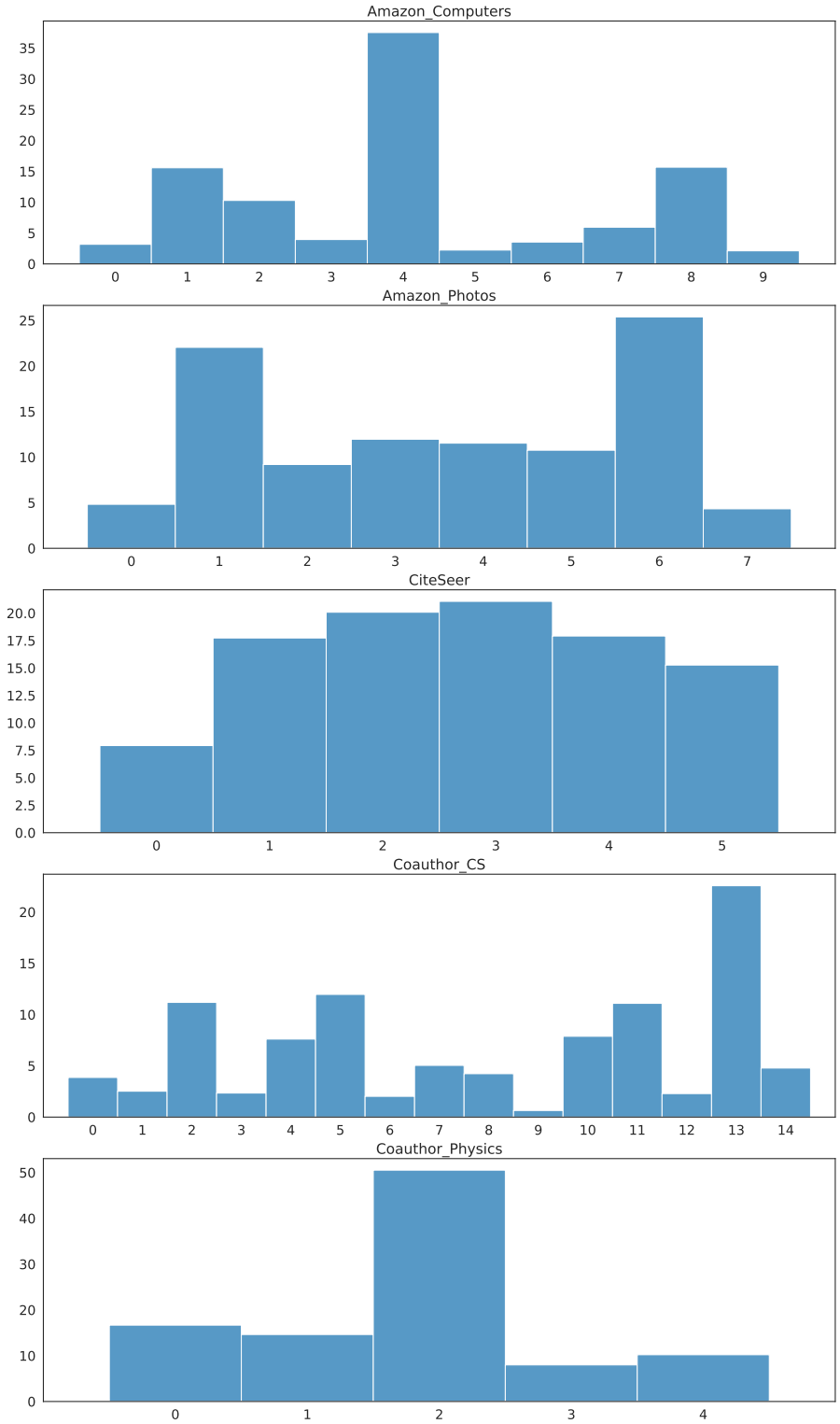


Figure D1: Plots of label distribution for each dataset. Each class for each dataset has at least one occurrence.

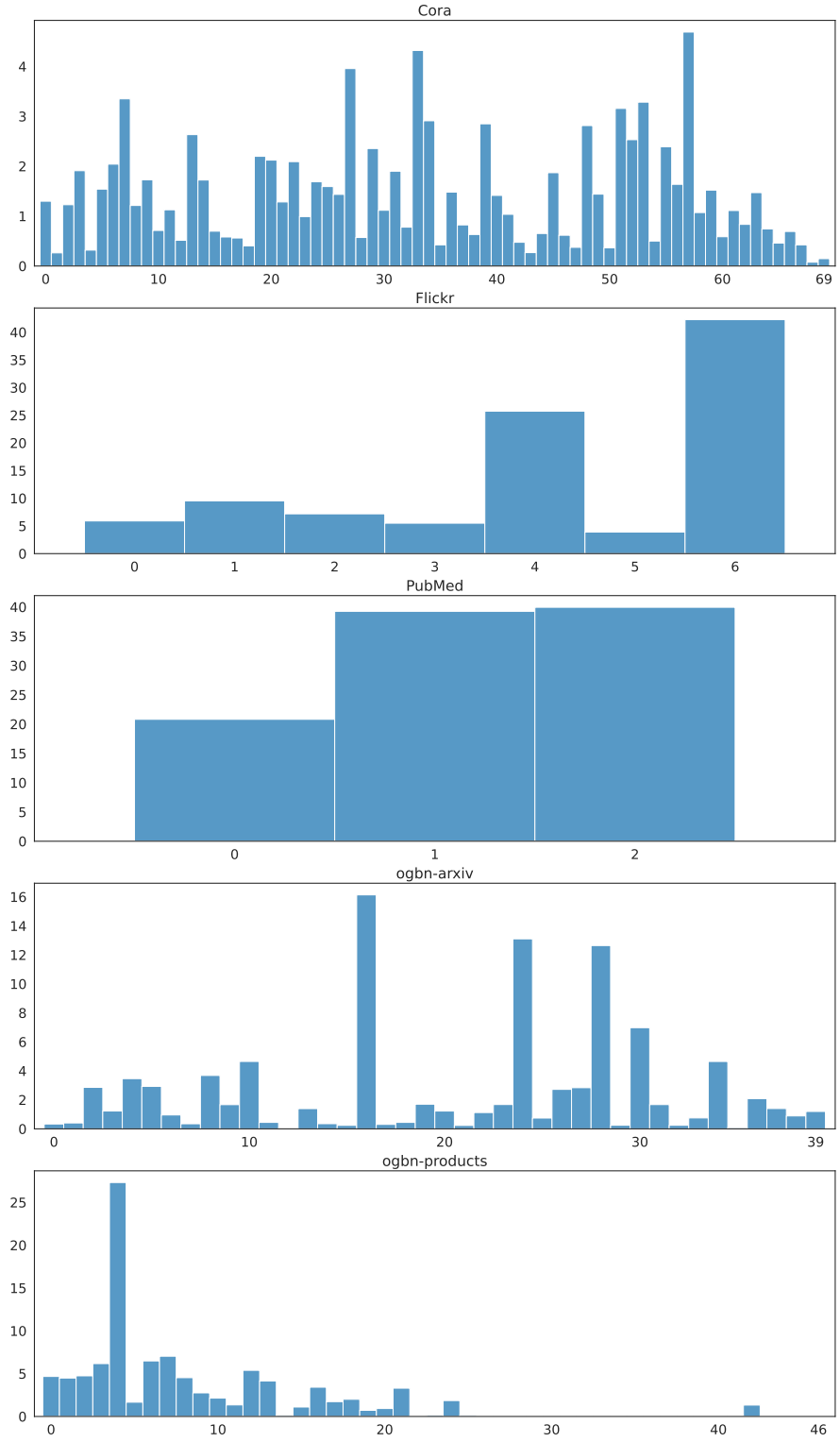


Figure D1: Plots of label distribution for each dataset. Each class for each dataset has at least one occurrence (cont.).

D.2 Hyperparameter Tuning

Hyperparameter tuning was done using Ray Tune (Liaw et al., 2018). The hyperparameters for the base model were tuned via random search using Table D3 for each model type (e.g. GCN, GAT, and GraphSAGE), for each dataset, except for the OGB (Hu et al., 2020) datasets, and each splitting scheme (FS vs LC and different value settings). For the OGB datasets, we took the hyperparameters and architectures from the corresponding leaderboard for all splitting schemes.

Once we got the best base model for each dataset and splitting scheme, we tune the hyperparameters for the CF-GNN model via random search using Table D4 for each model type (e.g. GCN, GAT, and GraphSAGE), for each dataset. For the FS splitting schemes, we set $\mathcal{D}_{\text{calib}} = \mathcal{D}_{\text{test}} = (1 - \mathcal{D}_{\text{train}} - \mathcal{D}_{\text{valid}})/2$. For the ogbn-products dataset, due to its size, we used a batch size of 512 for the CF-GNN training and also used a NeighborSampler with fanouts $[10, 10, 5]$ rather than a MultiLayerFullNeighborSampler.

[4]

All experiments with the ogbn-products datasets were run on a single A100 GPU while the remaining experiments for the other datasets were run on a single P100 GPU.

Table D3: Hyperparameter search space for the base GNN model for non-OGB datasets. The last two rows are layer-type specific for GAT and GraphSAGE, respectively.

Hyperparameter	Search Space
batch_size	64
lr	loguniform$(10^{-4}, 10^{-1})$
hidden_channels	$\{16, 32, 64, 128\}$
layers	$\{1, 2, 4\}$
dropout	uniform$(0.1, 0.8)$
heads	$\{2, 4, 8\}$
aggr_fn	{mean, gcn, pool, lstm}

Table D4: Hyperparameter search space for the CF-GNN model. The last two rows are layer-type specific for GAT and GraphSAGE, respectively.

Hyperparameter	Search Space
batch_size	64
lr	loguniform$(10^{-4}, 10^{-1})$
hidden_channels	$\{16, 32, 64, 128\}$
layers	$\{1, 2, 3, 4\}$
dropout	uniform$(0.1, 0.8)$
τ	loguniform$(10^{-3}, 10^{1})$
heads	$\{2, 4, 8\}$
aggr_fn	{mean, gcn, pool, lstm}

[4]For each dataset and split type, we provide the configuration files for the best corresponding base GNN and CFGNN architectures in the attached code.

E Additional Empirical Results, Analysis, and Insights

This section expands the figures and tables seen in the main body for all datasets considered. Figure E1 compares TPS and TPS-Classwise for all the datasets. We observe that TPS-Classwise achieves the desired label stratified coverage of 0.9 while TPS doesn't necessarily for the FS split. For the LC split, we observe that TPS-Classwise slightly improves on TPS in terms of label-stratified coverage; however, neither method necessarily achieves the target label-stratified coverage. In Figure E2, we find TPS-Classwise generally is less efficient than TPS for FS and LC splitting. Figure E3 compares the label stratified coverage for APS with and without randomization. We observe that randomization does sacrifice the label-stratified coverage for both FS and LC splitting. Noticeably, the change in coverage is smaller for the LC split. Figure E4 compares the efficiency of APS with and without randomization at $\alpha = 0.1$. For both split types (FS & LC), we observe that the randomized version of APS produces more efficient prediction sets, in line with Theorem 3.1. The efficiency improvements come with a sacrifice in label-stratified coverage since smaller prediction set sizes are preferred over covering every class, particularly if the classes are rare. To visualize this trade-off, we observe that in both Figure E1 and E3 the difference in label stratified coverage for ogbn-products with FS splitting is more extreme than with other datasets and with LC splitting. This is because ogbn-products has a lot of classes that have almost no representation (see Figure D1). Datasets that have near-uniform label distribution (e.g., PubMed, CiteSeer) – or when using LC split which controls for label counts – we observe label stratified coverage isn't sacrificed as much in the name of efficiency.

E.1 Conformalized GNN

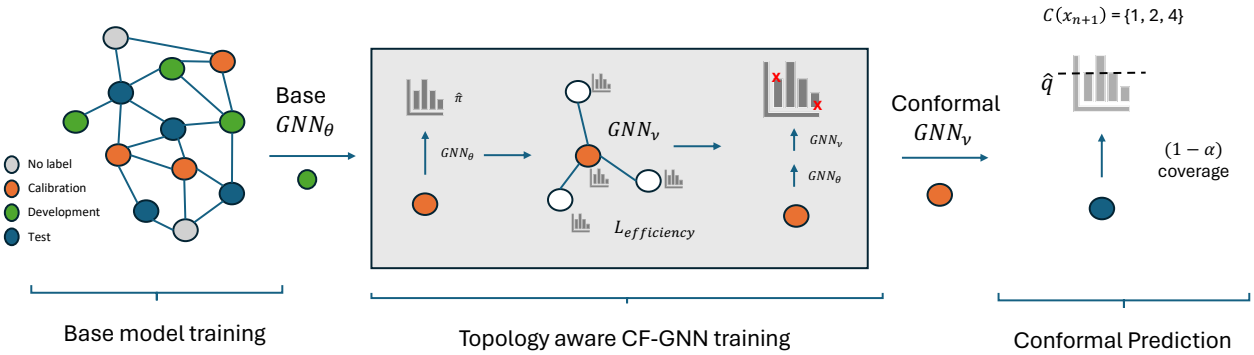


Figure E5: Procedure for training CFGNN. First (left), the base model is trained on the training set. Then, (middle) CFGNN is trained to maximize efficiency over the calibration set. Finally, (right) the non-conformity scores from the combined models are used to generate the prediction sets.

Section 3 introduces CFGNN (Huang et al., 2023) as a conformal prediction method specific for graphs. Figure E5 shows the end-to-end CFGNN procedure that is split into three steps. The first is to train a base GNN model, GNN_θ, on the development nodes, $\mathcal{V}_{\text{dev}}$, normally using a label-based loss function such as Cross Entropy Loss. Using the outputs of GNN_θ as inputs, a second GNN, GNN_φ is trained using an efficiency-based loss function proposed by Huang et al. (2023). Equation 22 presents the efficiency-based loss function for node classification, where σ is the sigmoid function and τ is a temperature hyperparameter. The calibration nodes, $\mathcal{V}_{\text{calib}}$, are split into two sets, $\mathcal{V}_{\text{cor-cal}}$ and $\mathcal{V}_{\text{cor-test}}$ for training and validation, respectively. The fully trained GNN_φ is then used for the conformal prediction and prediction set construction.

$$\begin{aligned} \hat{\eta} &= \text{DiffQuantile}(\{s(\boldsymbol{x}_i, y_i)\}, (1-\alpha)(1 + 1/|\mathcal{V}_{\text{cor-cal}}|)) \\ \mathcal{L}_{eff} &= \frac{1}{|\mathcal{V}_{\text{cor-cal}}|} \sum_{i \in \mathcal{V}_{\text{cor-cal}}} \sum_{k \in \mathcal{Y}} \sigma\left(\frac{s(\boldsymbol{x}_i, k) - \hat{\eta}}{\tau}\right) \end{aligned} \tag{22}$$

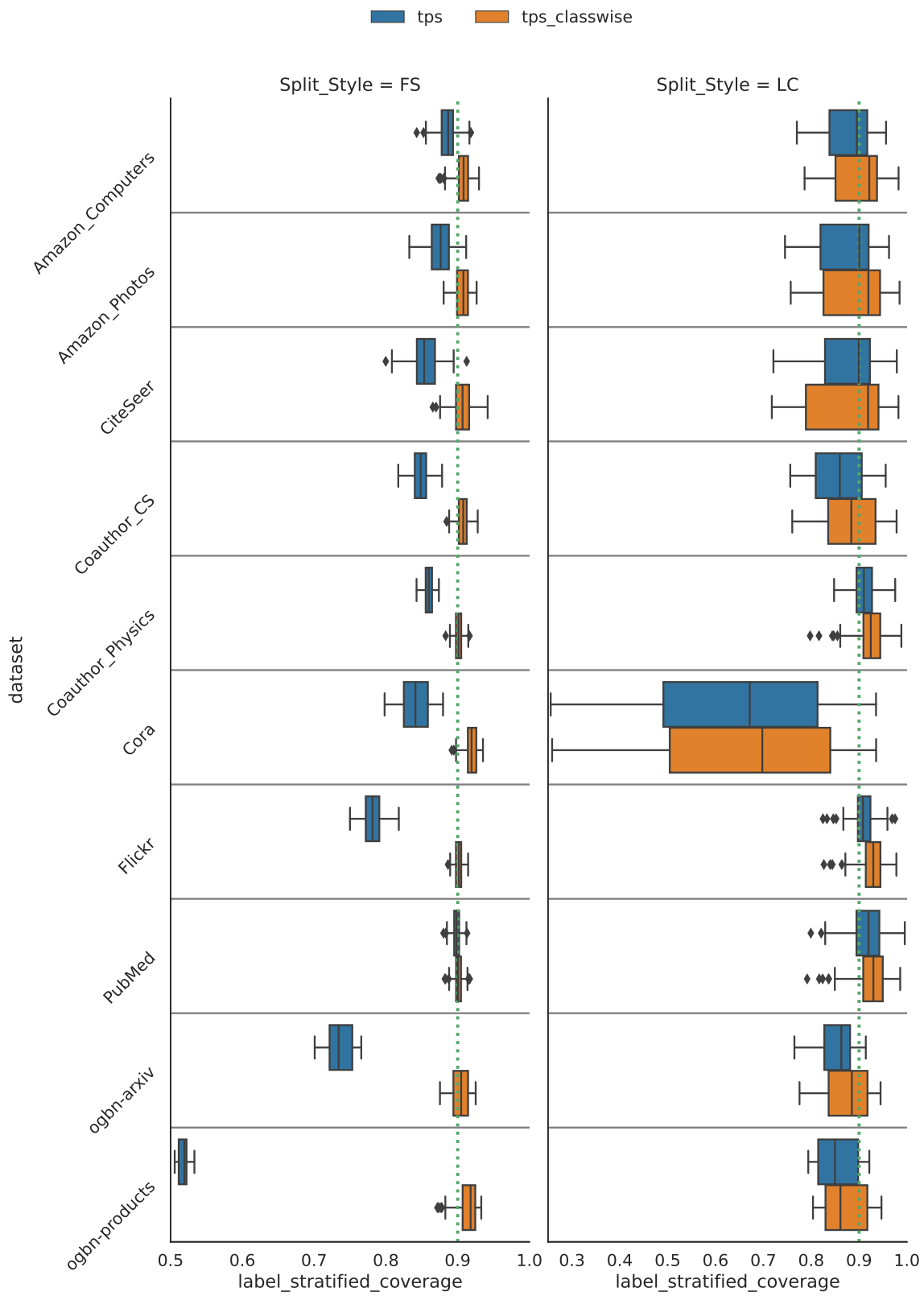


Figure E1: Plots for TPS vs TPS-Classwise for all the data sets at $1 - \alpha = 0.9$ coverage (green dotted line). TPS-Classwise, on average, meets label stratified coverage for FS split (left). For the LC split, the label stratified coverage slightly improves with TPS-Classwise but does not necessarily meet the $1 - \alpha$ coverage.

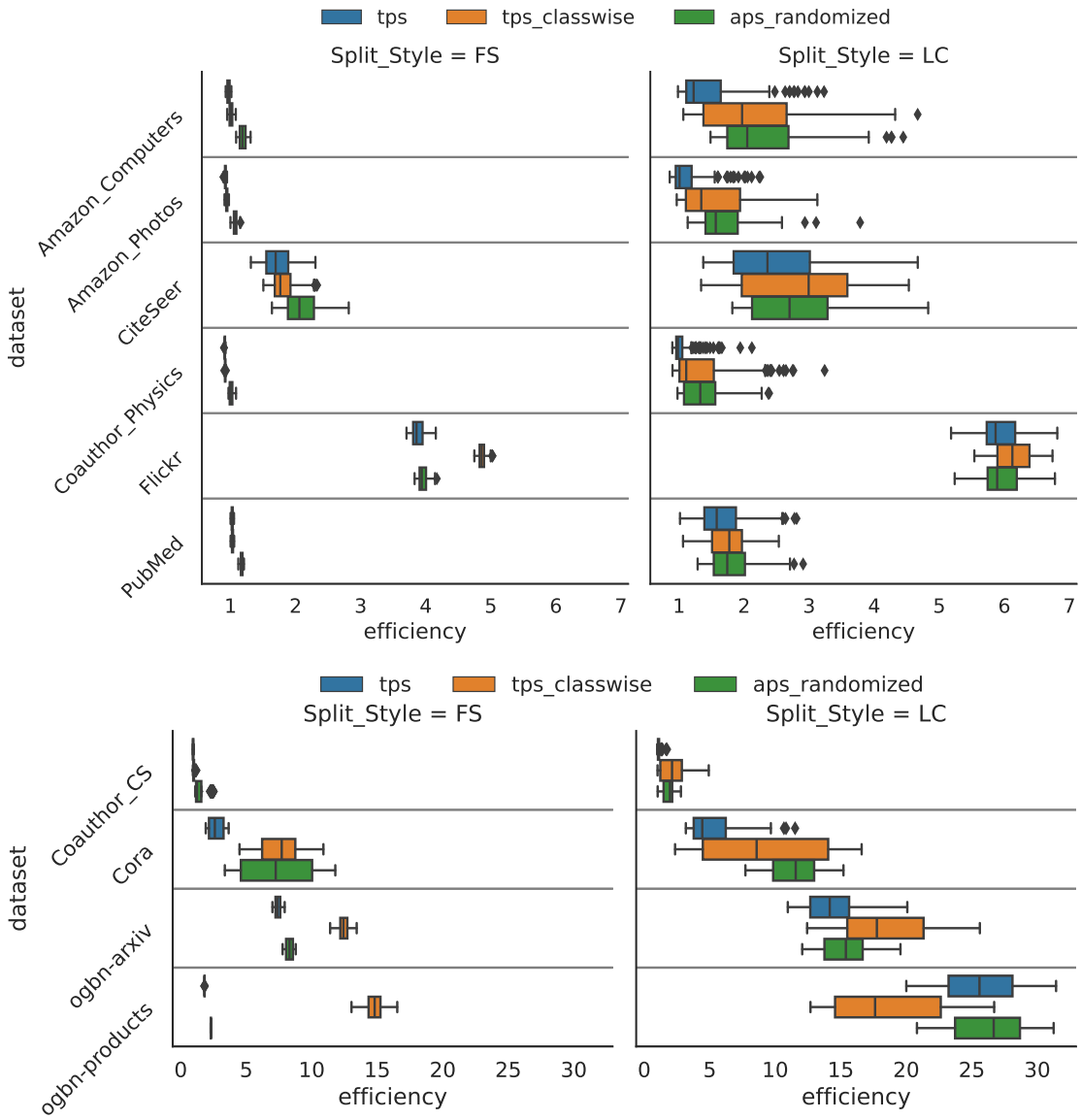


Figure E2: Plot for TPS vs TPS-classwise at $\alpha = 0.1$. For the FS split type, TPS-classwise becomes more inefficient compared to TPS for larger graph sizes but is competitive in other settings. To maintain label stratified coverage TPS-classwise may be forced to overcover certain classes at the cost of efficiency.

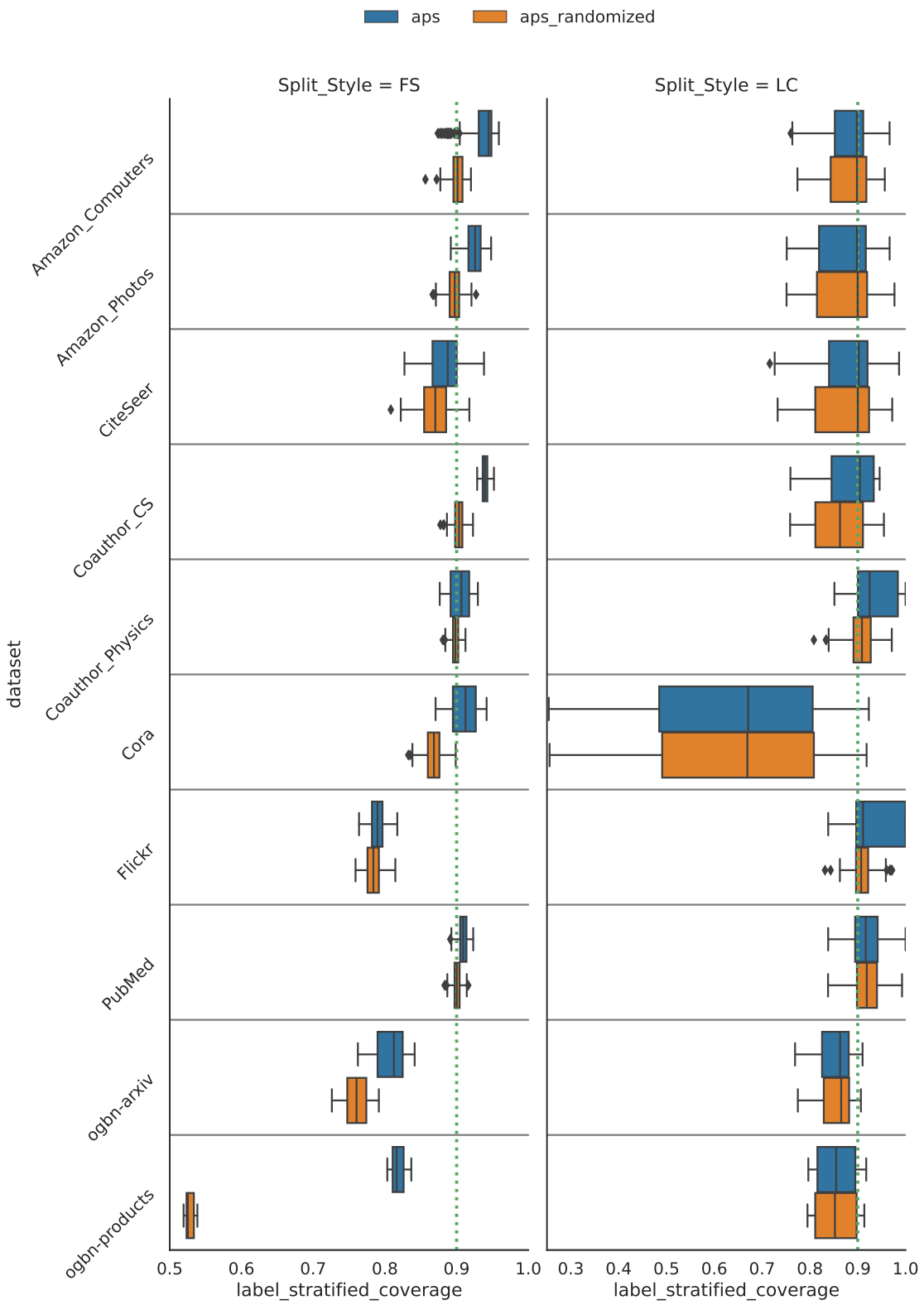


Figure E3: Plots for APS vs APS Randomized for all the data sets at $1 - \alpha = 0.9$ coverage (green dotted line). For the FS split type, APS-Randomized has a lower label stratified coverage. However, with the LC split type, the decrease in label-stratified coverage is not as significant.

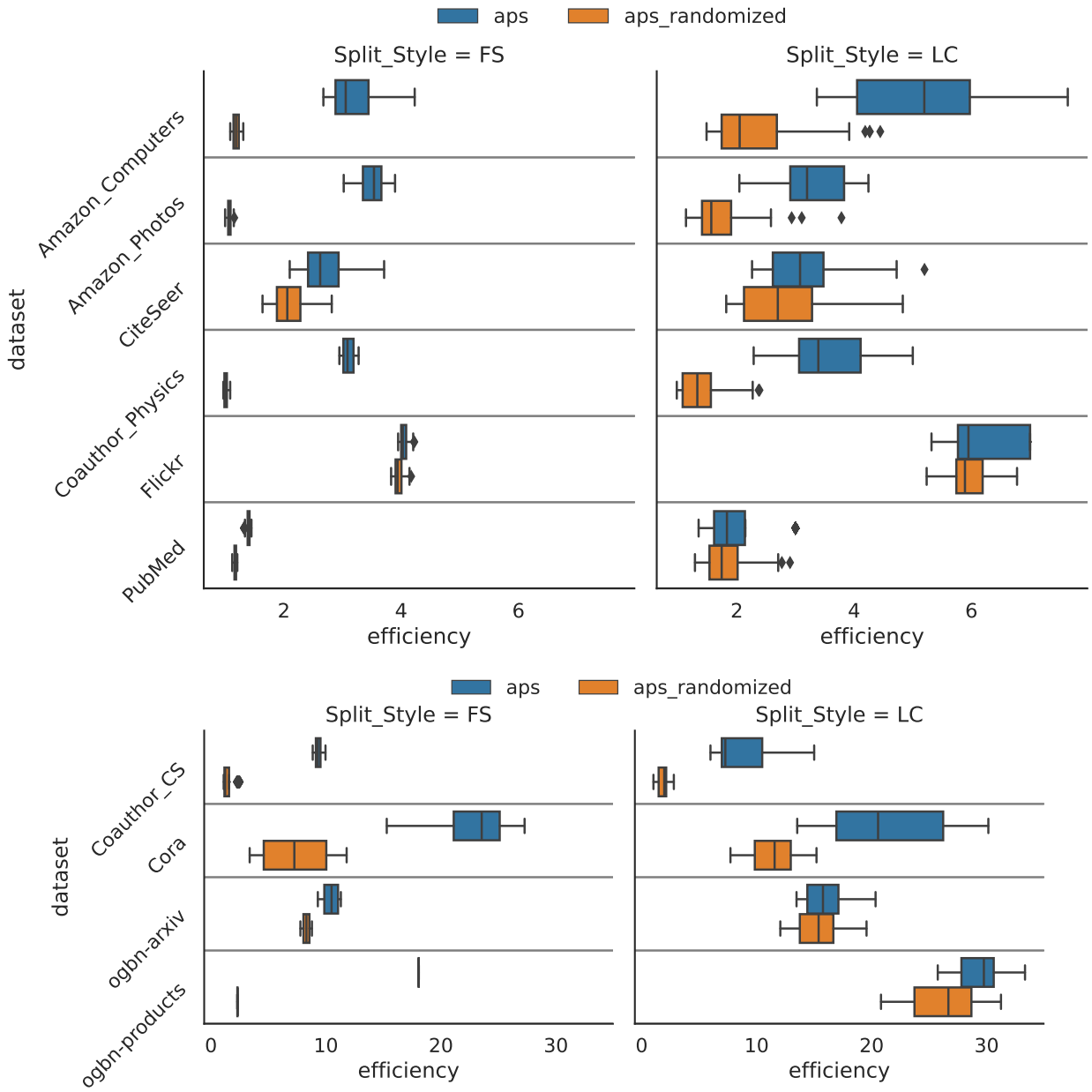
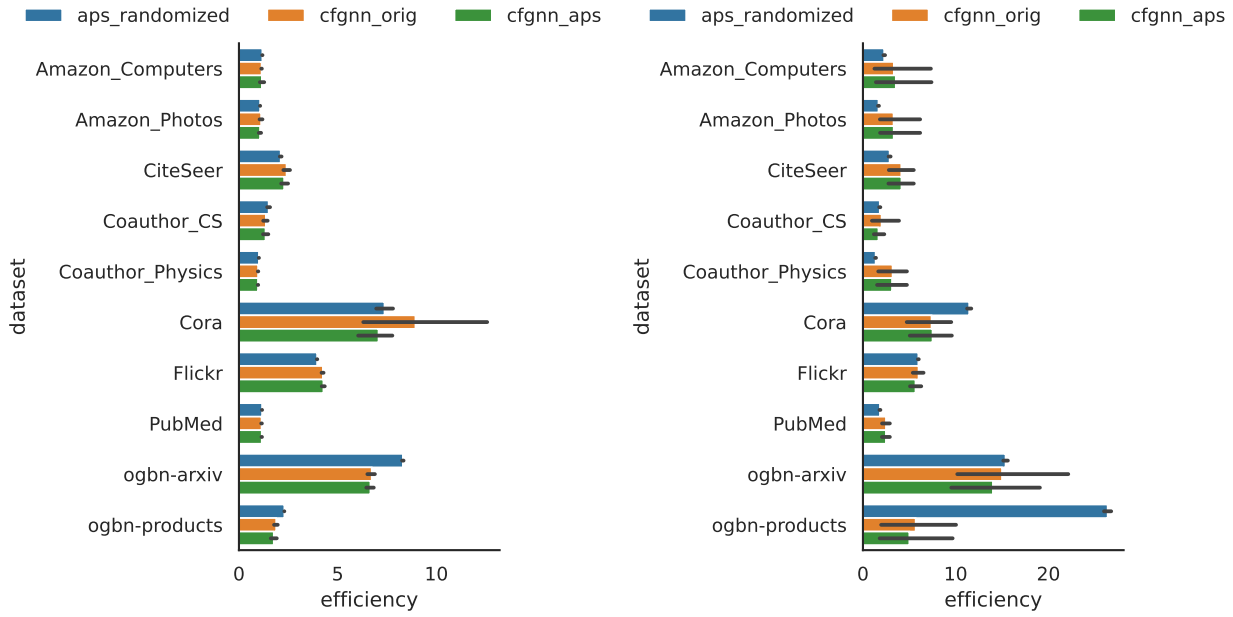


Figure E4: Plots for APS vs APS-randomized at $\alpha = 0.1$. For both split types (FS & LC), the randomized version of APS produces more efficient sets for all the datasets.

E.1.1 Further Discussions on CFGNNN

Impact of Inefficiency Loss: Figure E6 compares the efficiency of 'cfgnn_aps', 'cfgnn_orig,' and 'aps_randomized' for all the other datasets. For the FS split, we see that 'cfgnn_aps' improves upon 'cfgnn_orig' for all datasets and can improve upon 'aps_randomized' for most datasets. For the LC split, we see that CFGNN is still quite brittle for all datasets. For datasets with a larger number of classes, CFGNN can still improve upon 'aps_randomized' as seen with Cora and ogbn-products.



(a) FS split style

(b) LC split style

Figure E6: Bar charts denoting efficiency for 'cfgnn_aps', 'cfgnn_orig', and 'aps_randomized' for both split styles at $\alpha = 0.1$. We see that 'cfgnn_aps' improves or matches efficiency in most cases.

Scaling CFGNN: Expanding on Table 2 from Section 3, Table E1 present the runtime improvements (see Table E1a) as well as the efficiencies (see Table E1b) for each CFGNN implementation using the best CFGNN architecture found through hyperparameter tuning (see Section D4). We observe that our improved implementation achieves comparable efficiency to the original in only 50 epochs as opposed to 1000 used in the original.

Table E1: Impact of different CFGNN implementations starting from the baseline, then batching, and then both caching and batching combined. Used the **best** CFGNN architecture (w.r.t. validation efficiency) for each dataset. We run 5 trials for each setup and report a 95% confidence interval. OOM = Out of Memory

(a) Impact on Runtime

dataset(↓) / method(→)	original	batching	batching+caching
CiteSeer	379.28 ± 9.36	36.36 ± 0.97	27.45 ± 0.80
Amazon_Photos	496.36 ± 4.92	102.92 ± 0.95	54.44 ± 1.36
Amazon_Computers	664.98 ± 10.90	205.29 ± 3.96	73.85 ± 1.56
Cora	1378.01 ± 13.35	203.61 ± 2.52	73.60 ± 1.53
PubMed	571.60 ± 12.09	219.63 ± 5.23	109.68 ± 1.64
Coauthor_CS	638.56 ± 6.14	88.49 ± 1.14	31.67 ± 0.53
Coauthor_Physics	5942.30 ± 176.90	3918.38 ± 17.16	1585.26 ± 6.84
Flickr	868.87 ± 9.98	567.39 ± 6.50	56.71 ± 1.30
ogbn-arxiv	410.91 ± 8.29	373.19 ± 3.64	111.38 ± 1.95
ogbn-products	OOM	OOM	8709.16 ± 55.00

(b) Impact on Efficiency

dataset(↓) / method(→)	baseline	batching	batching+caching
CiteSeer	2.52 ± 0.27	2.45 ± 0.19	2.42 ± 0.37
Amazon_Photos	1.06 ± 0.01	1.06 ± 0.02	1.12 ± 0.02
Amazon_Computers	1.29 ± 0.05	1.15 ± 0.01	1.14 ± 0.01
Cora	6.81 ± 1.07	8.34 ± 1.12	7.96 ± 0.59
PubMed	1.17 ± 0.00	1.16 ± 0.00	1.17 ± 0.00
Coauthor_CS	1.10 ± 0.01	1.15 ± 0.01	1.14 ± 0.01
Coauthor_Physics	0.97 ± 0.01	1.01 ± 0.03	0.99 ± 0.01
Flickr	4.23 ± 0.07	4.23 ± 0.04	4.24 ± 0.03
ogbn-arxiv	7.07 ± 0.05	7.28 ± 0.06	6.91 ± 0.01
ogbn-products	OOM	OOM	1.86 ± 0.06

E.2 Overall Results

In Figure E7, we provide a plot of all the different methods discussed in this work for each dataset across different values of α. If applicable, for each method, we show the best-performing version, e.g., APS with randomization vs without and CFGNN with APS training ('cfgnn_aps') rather than TPS training ('cfgnn_orig'). We present the results for $k = 5$ for the different NAPS variations, since almost all datasets – except for CiteSeer and ogbn-arxiv – achieved their best efficiencies at or before that point.

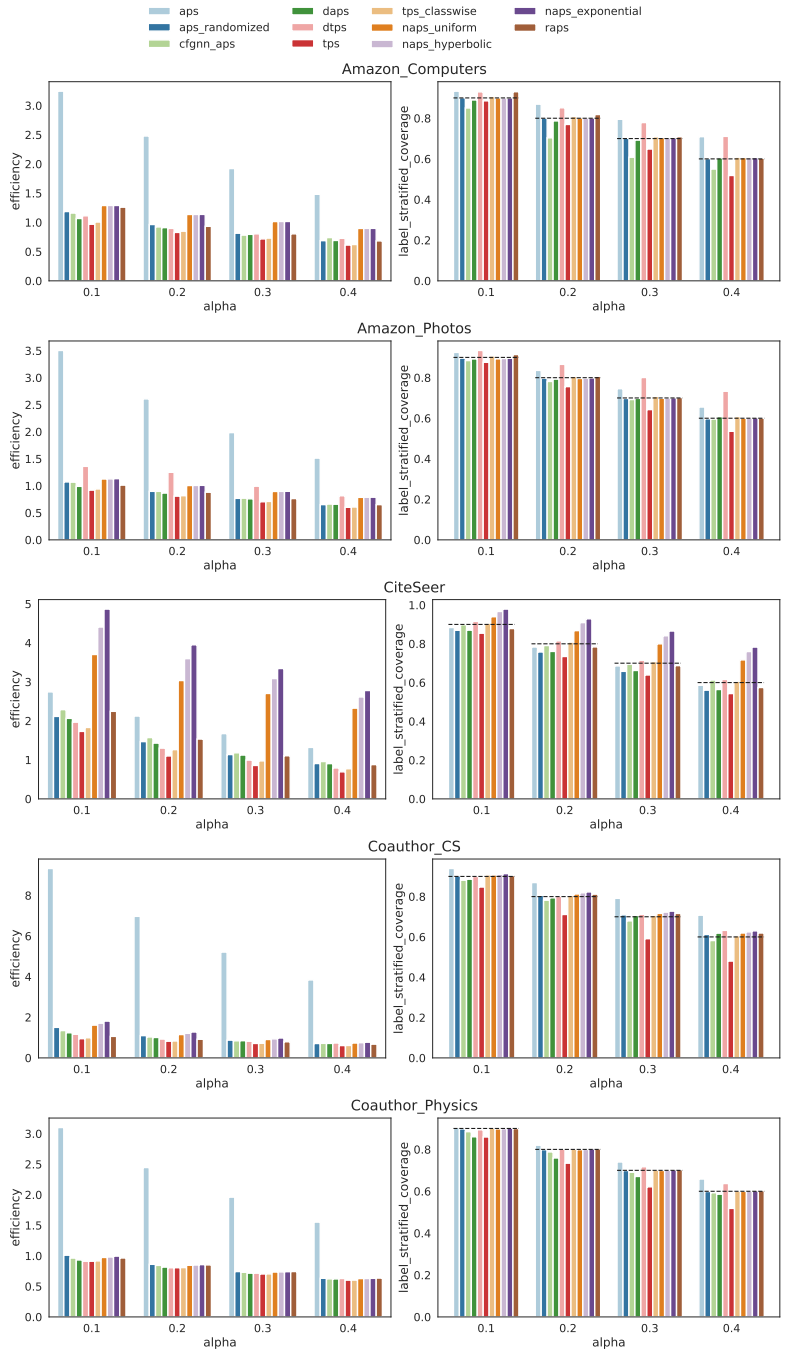


Figure E7: Plots for efficiency vs α for all the major methods (with best parameters) across all the datasets (cont.). Among the baseline methods, TPS consistently has the best efficiency. The results are for the FS split style. The dashed black line indicates the desired label stratified coverage.

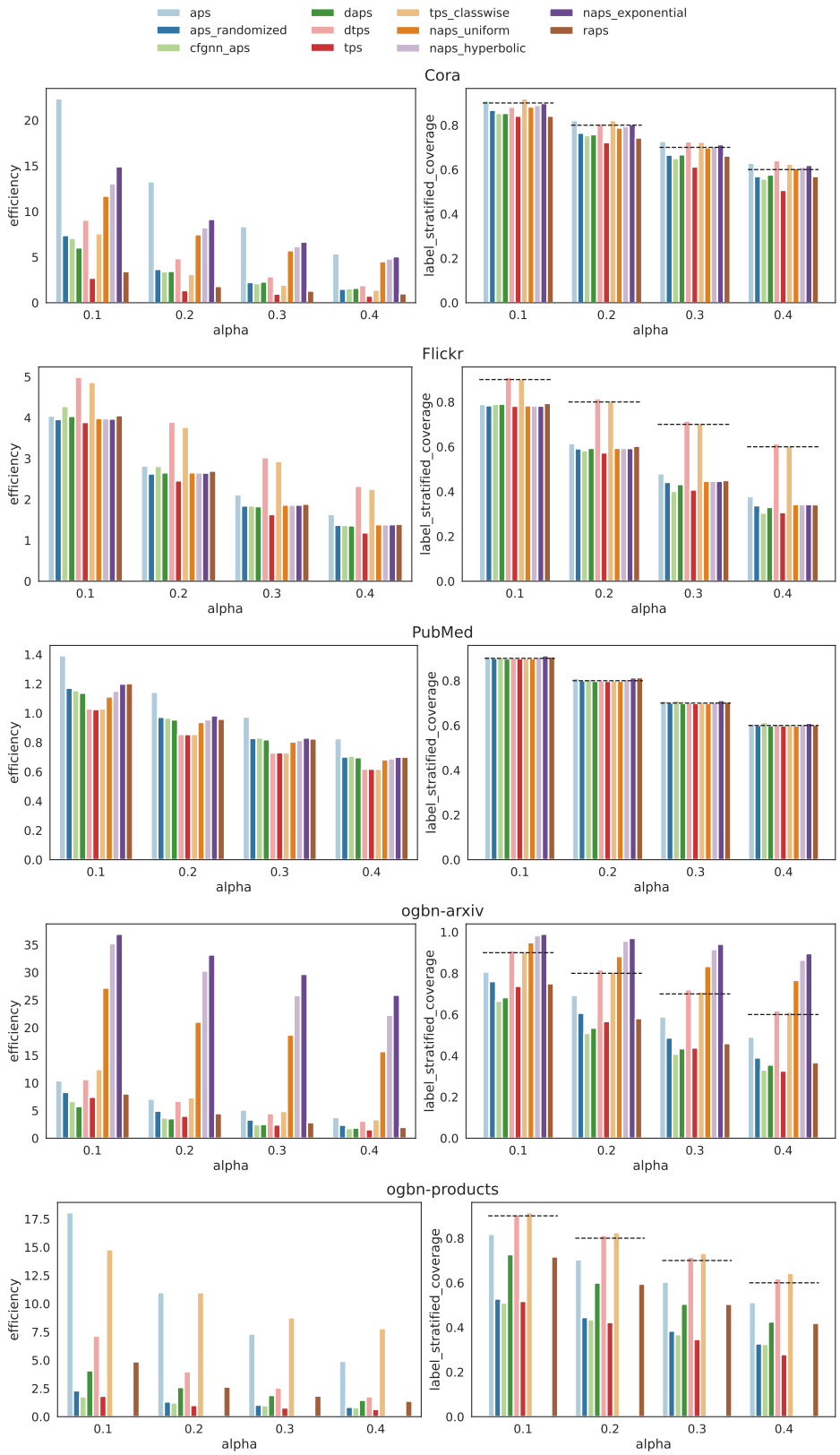


Figure E7: Plots for efficiency vs α for all the major methods (with best parameters) across all the datasets (cont.). Among the baseline methods, TPS consistently has the best efficiency. Results are for the FS split style. The dashed black line indicates the desired label stratified coverage. NAPS results for the ogbn-products dataset are omitted due to size and lack of data points with existing hardware.