Conformal Prediction and its application using Classification

Dhruv Parmar (202203008)
Mit Mehta (202203002)

Why Confidence Isn't Enough – The Case for Conformal Prediction

Problem Example (Skin Disease Classification)
Model Output (Softmax scores):

- Melanoma: 48%
- Benign nevus: 45%
- Others: 7%

$\rightarrow$ **Predicted label: Melanoma**
But confidence is only 48%. Can we trust this prediction?

What Actually Happens in Practice:
Across 100 predictions at 50% confidence:

- Correct: 58 times
- Incorrect: 42 times

Model is overconfident — 50% confidence 50% correctness!

Why Confidence Isn't Enough – Reasons

Why Does This Happen?

- Models are trained to minimize loss, not to produce truthful probabilities.
- Deep networks especially tend to be overconfident on unseen data.
- Regularization and data imbalance can further distort confidence.

On the test set, there's no guarantee that the predicted confidence actually means anything.

Conformal Prediction: A Statistically Valid Solution

Key Idea:

Instead of relying on raw model confidence, conformal prediction:

- Uses a calibration set (a portion of held-out data).
- Measures how often the model is wrong and how wrong it is.
- Builds a prediction set for each test point that contains the true label with high probability.

What is Conformal Prediction?

- **Definition & Purpose:** Conformal predictions provide a framework to generate prediction **intervals or sets** that come with a **formal guarantee on their reliability**.
- **Confidence Measures:** They assign a **confidence level** to each prediction, indicating the probability that the prediction interval or set contains the **true outcome**.
- **C**onformal prediction is a method that gives us prediction intervals or sets that come with a guarantee — like saying, "I'm 90% sure the real answer is inside this range."
  It works with any model and helps us know how confident we are in a prediction.

Basic Concepts for Conformal Prediction

Exchangeability

- Symmetric joint distribution
- Order statistics and uniform shuffling
- Self-sampling from empirical distribution

Conformal Scores

- Definition and purpose
- Types of scores:
  - Residual score
  - Scaled residual score
  - Conformalized Quantile Regression (CQR) score

Mathematical Tools

- Order statistics
- Empirical Cumulative Distribution Function (CDF)
- Quantiles of finite lists
- Behavior under transformations and permutations

Exchangeability — Definition and Formal View

- **Definition:** A random vector $(Z_1, Z_2, \ldots, Z_n)$ is said to be **exchangeable** if its distribution is **invariant under permutations**.
- **Formally:** For every permutation σ of indices $\{1, 2, \ldots, n\}$,

  $$(Z_1, \ldots, Z_n) \overset{d}{=} (Z_{\sigma(1)}, \ldots, Z_{\sigma(n)})$$

  where $\overset{d}{=}$ denotes equality in distribution.
- This concept extends to **infinite sequences**, meaning the property holds for all finite n.

Exchangeability — Implications and Examples

- **Implication:** Exchangeability implies that the elements are **identically distributed**, but **not necessarily independent**.
- It constrains the type of dependence: All permutations of the sequence are **equally likely**.
- **Examples of Exchangeable Sequences:**
  - I.I.D. sequences
  - Sampling without replacement
  - Structured dependence (e.g., urn models)

Exchangeability — Implications and Examples

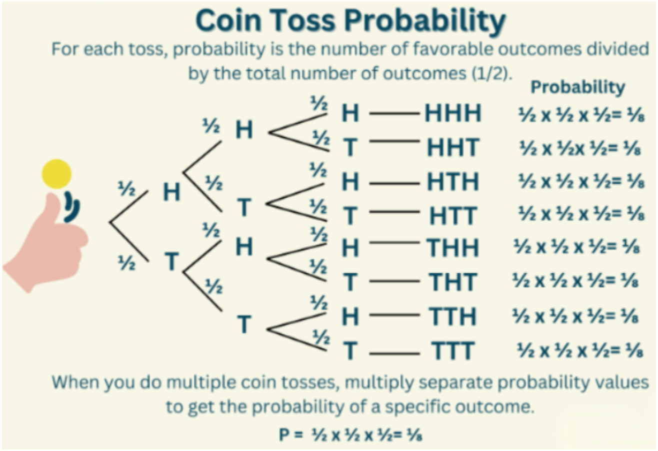

Coin Toss Probability
For each toss, probability is the number of favorable outcomes divided by the total number of outcomes (1/2).
Probability
½
H
T
HHH ½ x ½ x ½= ⅛
HHT ½ x ½x ½= ⅛
HTH ½ x ½ x ½= ⅛
HTT ½ x ½ x ½= ⅛
THH ½ x ½ x ½= ⅛
THT ½ x ½ x ½= ⅛
TTH ½ x ½ x ½= ⅛
TTT ½ x ½ x ½= ⅛
When you do multiple coin tosses, multiply separate probability values to get the probability of a specific outcome.
P = ½ x ½ x ½= ⅛

Figure: Outcomes of a coin tossed three times.

Exchangeability using an Example

- Each length-3 path has probability:

$$P(Z_1, Z_2, Z_3) = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$$

- For any permutation σ, the probability remains unchanged:

$$P(Z_1 = z_1, Z_2 = z_2, Z_3 = z_3) = P(Z_{\sigma(1)} = z_1, Z_{\sigma(2)} = z_2,$$

$\mathrm{Z}_{\sigma(3)} = z_3)$

- **Example:**

$$P(\{\mathrm{HHT}, \mathrm{HTH}, \mathrm{THH}\}) = 3 \times \left(\frac{1}{2}\right)^3 = \frac{3}{8}$$

- **Conclusion:** Only the **multi-set** of outcomes matters, not their order.

Exchangeability: Core Characterizations

- **Symmetric Joint Distribution:**
  - **Discrete:**

    $$p(z_1, \ldots, z_n) = p\left(z_{(\sigma(1))}, \ldots, z_{(\sigma(n))}\right) \text{ for all permutations } \sigma \in S_n$$

  - **Continuous:**

    $$f(z_1, \ldots, z_n) = f\left(z_{(\sigma(1))}, \ldots, z_{(\sigma(n))}\right) \text{ (almost everywhere)}$$

- **Given Order Statistics: Uniform Shuffling**
  Let $Z_{(1)} \leq \ldots \leq Z_{(n)}$ be the sorted values, then:
  $(Z_1, \ldots, Z_n) \mid (Z_{(1)}, \ldots, Z_{(n)}) \sim$ Uniform over all $n!$ permutations

  $$\text{Implication: } P(z_i \leq z_{(k)}) \geq \frac{k}{n}$$

- **Given Empirical Distribution: Self-Sampling**

  $$\hat{F}_n = \frac{1}{n} \sum_{i=1}^{n} \delta_{Z_i}$$

Purpose of Conformal Scores

- **Definition**: A **conformal score** $s(x, y)$ is a function that measures how unusual or nonconforming a data point (x, y) is relative to the training data.
- **Evaluate Fit**: The score helps assess how well a new example (x, y) aligns with trends in the training data.
- **Surprise Factor**: Higher values of $s(x, y)$ indicate the point is more surprising or less likely under the learned model.

Why It Matters

- **Core Concept**: Conformal scores are fundamental in constructing prediction sets $C(X_{n+1})$.
- **Prediction Sets**: The prediction set includes all y-values such that $s(X_{n+1}, y) \leq q$.
- **Valid Coverage**: This guarantees that the true value will fall within the set with high probability (e.g., 90%).

Types of Conformal Scores

Residual Score:

- **Formula:** $s(x, y) = |y - \hat{f}(x)|$
- Measures how far the actual value y is from the predicted value $\hat{f}(x)$.
- **Prediction Set:** Symmetric, fixed-width intervals around $\hat{f}(x)$.
- **Limitation:** Doesn't adapt to changing noise levels.

Scaled Residual Score:

- **Formula:** $s(x, y) = \frac{|y - \hat{f}(x)|}{\hat{\sigma}(x)}$
- Normalizes residual by dividing with estimated noise scale $\hat{\sigma}(x)$.
- **Prediction Set:** Adaptive intervals based on noise.
- **Benefit:** Handles heteroskedasticity (non-constant variance).

Types of Conformal Scores

CQR (Conformalized Quantile Regression) Score: Formula:

$$s(x, y) = \max\{\hat{\tau}(x; \alpha/2) - y,\ y - \hat{\tau}(x; 1 - \alpha/2)\}$$

- Uses quantile regression to measure deviation from prediction intervals.
- **Prediction Set:** Asymmetric intervals, expand or shrink with data.
- **Benefit:** Great for skewed or asymmetric distributions.

Order Statistics

Definition: Order statistics refer to the values of a list arranged in non-decreasing order. The k-th order statistic is the k-th smallest value in the list.

Formal Definition: Given a list $z = (z_1, z_2, \dots, z_n)$, the k-th order statistic is:

$$z_{(k)} = \inf \left\{ v \in \mathbb{R} : \sum_{i=1}^{n} \mathbf{1}\{z_i \leq v\} \geq k \right\}$$

Example: $z = (3, 2, 1, 2)$ Rearranged: $(1, 2, 2, 3)$

- $z_{(1)} = 1$
- $z_{(2)} = 2$
- $z_{(3)} = 2$
- $z_{(4)} = 3$

Order statistics are always uniquely defined, even if values repeat.

Empirical CDF of a Finite List

Definition: The empirical CDF for a finite list tells us the fraction of values $\leq v$.

Formal Definition:

$$F_z(v) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{z_i \leq v\}$$

Example: $z = (3, 2, 1, 2)$, $n = 4$

- $F_z(1) = \frac{1}{4}$ (only 1 value ≤ 1)
- $F_z(2) = \frac{3}{4}$ (three values ≤ 2)
- $F_z(3) = \frac{4}{4} = 1$

The CDF increases with v and always ends at 1.

Quantile of a Finite List

Definition: The τ-quantile is the smallest value such that at least τ fraction of the data lies at or below it.

Formal Definition:

$$\text{Quantile}(z; \tau) = \inf \{v \in \mathbb{R} : F_z(v) \geq \tau\}$$

Example: $z = (3, 2, 1, 2)$ From previous slide:

- $F_z(1) = 0.25$
- $F_z(2) = 0.75$
- $F_z(3) = 1$

Quantiles:

- $\tau = 0.25 \Rightarrow \text{Quantile} = 1$
- $\tau = 0.5 \Rightarrow \text{Quantile} = 2$
- $\tau = 1 \Rightarrow \text{Quantile} = 3$

Quantiles are found by inverting the CDF.

What Happens Under Transformations

Monotonic Transformations (e.g., $f(x) = 2x + 1$)
If f is increasing, then:

$$f(z_{(k)}) = (f(z))_{(k)}$$

$$\text{Quantile}(f(z); \tau) = f(\text{Quantile}(z; \tau))$$

$$F_{f(z)}(v) = F_z(f^{-1}(v))$$

Example: $z = (3, 2, 1, 2)$, $f(x) = x + 1$ Then: $f(z) = (4, 3, 2, 3)$

- $\text{Quantile}(z; 0.5) = 2$
- $\text{Quantile}(f(z); 0.5) = 3 = f(2)$

Permutations
Shuffling elements of z does *not* change order statistics, quantiles, or CDF.
Example: Permuted $z = (2, 3, 2, 1)$ Sorted: $(1, 2, 2, 3)$ All statistics remain unchanged.

Properties Under Exchangeability

The deterministic relationships between order statistics, quantiles, and CDFs extend to probabilistic bounds under **exchangeability**.

Assume $Z \in \mathbb{R}^n$ is exchangeable and fix any $i \in [n]$.

1. For any $k \in [n]$,
$$\mathbb{P}(Z_i \leq Z_{(k)}) \geq \frac{k}{n} \quad \text{and} \quad \mathbb{P}(Z_i < Z_{(k)}) \leq \frac{k-1}{n}$$

2. For all $\tau \in [0,1]$,
$$\mathbb{P}(Z_i \leq \text{Quantile}(Z;\tau)) \geq \tau$$
and, if $\tau > 0$,
$$\mathbb{P}(Z_i < \text{Quantile}(Z;\tau)) < \tau$$

3. For all $\tau \in [0,1]$,
$$\mathbb{P}(F_Z(Z_i) \leq \tau) \leq \tau \quad \text{and} \quad \mathbb{P}(F_Z(Z_i) \geq \tau) \geq 1-\tau$$

Properties Under Exchangeability

If the elements of Z are almost surely distinct, then:

4. For any $k \in [n]$,
$$\mathbb{P}(Z_i \leq Z_{(k)}) = \frac{k}{n}$$
5. For all $\tau \in [0,1]$,
$$\mathbb{P}(Z_i \leq \text{Quantile}(Z;\tau)) = \frac{\lceil n\tau \rceil}{n}$$
6. For all $\tau \in [0,1]$,
$$\mathbb{P}(F_Z(Z_i) \leq \tau) = \frac{\lfloor n\tau \rfloor}{n}$$

This fact underlies the conformal p-value result:
$$p = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{Z_i \geq Z_n\} \quad \text{satisfies} \quad \mathbb{P}(p \leq \tau) \leq \tau$$
by applying (i) with $i = n$ and $k = \lceil (1-\tau)n \rceil$.

Conformal Prediction: Key Concepts

- **Data Sequence & Prediction Target**
  - Begin with an exchangeable sequence:

$$(X_1, Y_1), \ldots, (X_n, Y_n), (X_{n+1}, Y_{n+1})$$

  - X_i (feature) and Y_i (response) for $i = 1, \ldots, n$
  - Y_{n+1} is **unobserved** and is the prediction target given X_{n+1}
- **Prediction Set Construction**
  - Construct a set $C(X_{n+1}) \subseteq \mathcal{Y}$ satisfying:

$$P(Y_{n+1} \in C(X_{n+1})) \geq 1 - \alpha$$

  - Guarantees marginal predictive coverage at level $1 - \alpha$
- **Score Function** $s((x, y); D)$
  - Maps a data point and dataset D to a real number
  - *Interpreted as an error measurement:*

$$s((x, y); D) = |y - \hat{f}(x; D)|$$

  - High score $\rightarrow$ large discrepancy between actual and predicted values

Conformal Prediction: Key Concepts

- **Symmetric Score Function**
  - Invariance under data permutation:

$$s((x,y);D) = s((x,y);D^{\sigma})$$

  - Ensures that the model's training process does not depend on the order of data points

Example of Symmetric Score Function

Dataset Setup

Consider the dataset of responses:

$$D = \{3, 8, 12, 15, 19, 25, 30, 33, 40\}.$$

The ordering of the elements in D can be arbitrarily permuted.

Defining the Score Function

Define the score function based on the *median* of D:

$$s(y; D) = |y - \text{median}(D)| .$$

Computing the Median

After sorting, the dataset remains:

$$D = \{3, 8, 12, 15, 19, 25, 30, 33, 40\}.$$

Example of Symmetric Score Function

Computing the Median

With 9 elements, the median (the 5th element) is:

$$\text{median}(D) = 19.$$

Evaluating the Score Function

For a test value $y = 27$:

$$s(27; D) = |27 - 19| = 8.$$

Symmetry Property

Since the median is invariant under any permutation of D, the score function

$$s(y; D) = |y - \text{median}(D)|$$

is symmetric.

What is Marginal Coverage in Conformal Prediction?

- **Marginal Coverage:** A guarantee that the true label Y_{n+1} for a new test point lies within the conformal prediction set $C(X_{n+1})$ with high probability ($\geq 1 - \alpha$).
  - If the data $(X_1, Y_1), \dots, (X_{n+1}, Y_{n+1})$ are exchangeable,
  - The score function s is symmetric,

  then

$$\mathbb{P}(Y_{n+1} \in C(X_{n+1})) \geq 1 - \alpha$$

- **Interpretation:** Over many test points, the prediction set includes the true label with at least $(1 - \alpha) \times 100\%$ probability.

Why $1-\alpha$ Appears in Conformal Prediction

Confidence Level Specification

Conformal prediction is designed to guarantee that you achieve the confidence level you specify.
You choose the desired confidence level by setting α (e.g., for 95% confidence, set $\alpha = 0.05$).

Internal Threshold via Quantiles

The method uses the chosen α value to set an internal threshold:

$$\text{Threshold} = (1\text{-}\alpha) \text{ quantile of the scores.}$$

This quantile is the score value below which $(1-\alpha)$ (e.g., 95%) of all scores fall.

Guarantee via Exchangeability

Owing to the symmetry/exchangeability assumption, the test score S_{n+1} satisfies:

$$P\left(S_{n+1} \leq (1\text{-}\alpha) \text{ quantile}\right) \geq 1-\alpha,$$

ensuring that the prediction set covers the true response with probability at least $1-\alpha$.

How is Marginal Coverage Achieved?

- For a new test point X_{n+1}, construct a prediction set:

$$C(X_{n+1}) = \left\{ y : S^{y}_{n+1} \leq \hat{q}_y \right\}$$

  where S^{y}_{n+1} is the score of label y on the test point, and $\hat{q}_y$ is a quantile of the training scores.
- Specifically, $\hat{q}_y$ is the $(1-\alpha)(1+\frac{1}{n})$th-quantile of the training scores.
- **Goal:** Only include labels y that are consistent with trends observed in the training data.
- **Example:** If $\alpha = 0.1$, the prediction set covers the true label in at least 90 out of 100 test cases.

Full Conformal Prediction Algorithm – Part 1

Input:

- Training data: $D_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$
- Test input: X_{n+1}
- Score function: $s((X, Y); D)$
- Significance level: $\alpha \in (0, 1)$

Steps:

1. **Loop over candidate labels:** For each $y \in \mathcal{Y}$ (e.g., $y = 0, 1, \ldots, 9$):
2. **Augment dataset:**

$$D_{n+1}^{y} = D_n \cup \{(X_{n+1}, y)\}$$

3. **Compute conformity scores using** D_{n+1}^{y}:
   - For training: $S_i^y = s((X_i, Y_i); D_{n+1}^y), \quad i = 1, \ldots, n$
   - For test: $S_{n+1}^y = s((X_{n+1}, y); D_{n+1}^y)$

Full Conformal Prediction Algorithm – Part 2

4. **Compute conformal quantile:**

$$\hat{q}_y = \text{Quantile}(S_1^y, \ldots, S_n^y; (1-\alpha)(1+\frac{1}{n}))$$

5. **Include conforming values:** If

$$S_{n+1}^y \leq \hat{q}_y$$

then include y in prediction set:

$$C(X_{n+1}) \leftarrow C(X_{n+1}) \cup \{y\}$$

6. **Return prediction set:**

$$C(X_{n+1}) = \{y : S_{n+1}^y \leq \hat{q}_y\}$$

Coverage Guarantee :

- If the data is exchangeable and the score function is symmetric:

$$\mathbb{P}(Y_{n+1} \in C(X_{n+1})) \geq 1-\alpha$$

The Problem: Why Simple Doesn't Work

- **Goal:** Build a prediction set $C(X_{n+1})$ for a new point X_{n+1} that contains the true Y_{n+1} with probability $1 - \alpha$ (e.g., 95%).
- **Naive Idea:**
  - Train a model on data D_n.
  - Compute nonconformity ("weirdness") scores for each training point.
  - Choose the 95th percentile score as a threshold.
  - For a new X_{n+1}, include a candidate y if its score is below this threshold.
- **The Catch:** Overfitting causes training points to have lower scores, while the new test point likely has a higher score. This leads to a threshold that is too low, resulting in less than 95% coverage.

The Conformal “Trick”: Treating Everyone Equally

- **Core Insight:** The test point is treated differently from training points.
- **Full Conformal’s Idea:**
  - **Assume** we know the true Y_{n+1} and form the augmented dataset $D_{n+1} = D_n \cup \{(X_{n+1}, Y_{n+1})\}$.
  - Train the model on all $n + 1$ points.
  - Compute scores $S_1, S_2, \ldots, S_{n+1}$ using the same procedure for every point.
- **Result:** The new point is now treated identically to the training points.

Symmetry $\Longrightarrow$ Exchangeability

- **Symmetry:** All scores $S_1, \ldots, S_{n+1}$ are computed in the same way.
- **Exchangeability:**
  - The joint distribution of the scores is invariant to the order of the data.
  - Analogous to drawing $n+1$ balls from an urn where every ordering is equally likely.

Exchangeability Guarantees Coverage

- **Key Property:** For exchangeable scores, the rank of any individual score (e.g., S_{n+1}) is random.
- **Quantile Rule:**
  - Compute the $(1-\alpha)$ quantile of $\{S_1, \dots, S_{n+1}\}$.
  - S_{n+1} falls below this quantile with probability at least $1-\alpha$.
- This quantile check is the backbone of the conformal prediction guarantee.

Putting It All Together - Why It Works

- **Problem:** Naive scores are biased due to overfitting.
- **Conformal Fix:** By hypothetically including Y_{n+1}, all scores are computed symmetrically.
- **Outcome:** This symmetry means the scores are exchangeable.
- **Magic Property:** For exchangeable scores, the chance that S_{n+1} is below the $(1-\alpha)$ quantile is at least $1-\alpha$.
- **Conclusion:** Hence, the true Y_{n+1} is included in $C(X_{n+1})$ with probability at least $1-\alpha$.

But We Don't Know Y_{n+1}?

- Since Y_{n+1} is unknown, the algorithm tests every candidate y.
- For each candidate y:
  - Form the augmented dataset $D_n \cup \{(X_{n+1}, y)\}$.
  - Train the model and compute scores $S_1^y, \ldots, S_{n+1}^y$.
  - Include y in the prediction set if S_{n+1}^y is below the threshold.
- **Coverage Guarantee:** When $y = Y_{n+1}$, the exchangeability of scores ensures coverage $\geq 1 - \alpha$.

Split Conformal Prediction: Overview & Key Differences

What is Split Conformal Prediction?

- A variant of conformal prediction using a **data split** to simplify computation.
- Avoids retraining the model for every possible response y.

Data Splitting Strategy:

- **Pretraining Set (D_{pre}):** Used solely for training the predictive model.
- **Calibration Set (D_n):** Used exclusively to calibrate the score function.

Comparison with Full Conformal Prediction:

- **Full Conformal:**
  - Uses all data for both training and calibration.
  - Requires a symmetric score function.
  - Retrains/evaluates the model for each possible y.
- **Split Conformal:**
  - Uses disjoint datasets for training and calibration.
  - Works with any pretrained score function without retraining.

Algorithm: How It Works

Step 1: Construct the Score Function

- Train a model on D_{pre} to build the score function $s(x, y)$.
- *Example (residual score):* $s(x, y) = |y - \hat{f}(x; D_{\text{pre}})|$.

Step 2: Calibration

- Compute scores $S_i = s(X_i, Y_i)$ for each calibration data point.

Step 3: Determine the Threshold

- Calculate the quantile:

$$\hat{q} = \text{Quantile}(S_1, \ldots, S_n; (1 - \alpha)(1 + 1/n))$$

Step 4: Form the Prediction Set

- For a new test point X_{n+1}, the prediction set is:

$$C(X_{n+1}) = \{y \in Y : s(X_{n+1}, y) \leq \hat{q}\}$$

Key Benefits:

- **Simplicity & Efficiency:** One model fit required.
- **Theoretical Guarantee:** Inherits the marginal coverage guarantee from the full conformal method.

Classification - Picking the Best Guess

Classification Model

Output Label: "CAT"

- We train a model (e.g., neural network, logistic regression) to predict a category.
- **Example:** Given a picture, the model outputs "CAT".
- The model usually outputs probabilities (e.g., Cat: 70%, Dog: 25%, Bird: 5%) and picks the highest.
- **Problem:** How sure are we? If the probabilities are close (e.g., Cat: 51%, Dog: 49%), we lack reliable confidence.

Conformal Idea - Let's Add Confidence!

Calibration Set → Model Scores → List of "Weirdness" Scores

Goal: Create a prediction set (e.g., {**CAT**} or {**CAT**, **DOG**}) that contains the true answer 95% of the time.

- **How?** Use a calibration set — data unseen during training with known true labels.
- **Calculate "Weirdness" Scores:** For each calibration item, compute how "weird" the true label is.
- **Simple Idea:** Score $= 1 -$ Probability (TRUE label)
- **Example:**
  - If the true label is DOG and $P(\text{DOG}) = 10\%$, then Score $= 1 - 0.10 = 0.90$ (High = Weird).
  - If $P(\text{DOG}) = 98\%$, then Score $= 1 - 0.98 = 0.02$ (Low = Not Weird).

Finding the "Confidence Threshold"

Histogram of "Weirdness" Scores

Threshold: Cutting off the top 5% highest scores (labeled q)

- **Collect Scores:** Gather all the weirdness scores from the calibration set.
- **Set Confidence Level:** For 95% confidence, allow a 5% error ($\alpha = 0.05$).
- **Find the Threshold:** Determine the score value that includes 95% of the calibration scores below it.
- **Interpretation:** This threshold q tells us "how weird is too weird" to maintain 95% confidence.

Making Confident Predictions!

New Picture $\rightarrow$ **Model** $\rightarrow$ **Probabilities**

Comparison Box $\rightarrow$ **Output Set:** $\{$**CAT**$\}$

- **New Picture:** The model outputs probabilities (e.g., Cat: 70%, Dog: 25%, Bird: 5%).
- **Calculate Weirdness:**
  - $\text{Weirdness}(\text{CAT}) = 1 - 0.70 = 0.30$
  - $\text{Weirdness}(\text{DOG}) = 1 - 0.25 = 0.75$
  - $\text{Weirdness}(\text{BIRD}) = 1 - 0.05 = 0.95$
- **Compare to Threshold:** Include classes with weirdness scores $\leq q$.
  - If $q = 0.80$, then CAT and DOG are included, while BIRD is excluded.
- **Final Prediction Set:** $\{\text{CAT}, \text{DOG}\}$
- **Guarantee:** This set contains the true answer at least 95% of the time.

CIFAR-10 Classification using MAPIE

- **Dataset:** CIFAR-10 with 10 classes:
  - Horse, Dog, Cat, Frog, Deer, Bird, Airplane, Truck, Ship, Automobile
- **Objective:** Compare prediction sets estimated by different conformal methods using `MapieClassifier`.
- **Model:** Train a small Convolutional Neural Network (CNN) for image classification.
- **Compatibility:** Create custom class `TensorflowToMapie` to bridge TensorFlow and MAPIE.

Data Splitting and Usage in CIFAR-10 Classification

1. **Dataset Source:** CIFAR-10 dataset is downloaded from the `TensorFlow Datasets` library.
2. **Data Splitting:** The original training set is divided into:
   - **Training set** – for training the neural network.
   - **Validation set** – for monitoring overfitting during training.
   - **Calibration set** – for calibrating conformal scores in `MapieClassifier`.

Create Calibration Set

- Start with the full dataset: **1000 images (X), 1000 labels (y)**.
- **Calibration Set:**
  - 10% of total = 100 samples ⇒ `X_calib, y_calib`
  - Used for confidence calibration (e.g., conformal prediction)
- **Remaining Set:**
  - 90% of total = 900 samples ⇒ `X_train_temp, y_train_temp`
- Visual:

$$[\text{Total Data}] \rightarrow [\text{Calibration (10\%)}] + [\text{Remaining (90\%)}]$$

Split Remaining into Training and Validation

- Input: **X_train_temp, y_train_temp** (900 samples)
- **Validation Set:**
  - 33% of 900 = 297 samples $\Rightarrow$ `X_val, y_val`
  - Used for hyperparameter tuning and early stopping
- **Training Set:**
  - Remaining 67% = 603 samples $\Rightarrow$ `X_train, y_train`
  - Used for model learning
- Visual:

$$\text{[Remaining 90\%]} \rightarrow \text{[Validation (33\%)]} + \text{[Training (67\%)]}$$

Final Output Key Points

- Final Sets Returned:
  - `X_train, y_train` (603 samples)
  - `X_val, y_val` (297 samples)
  - `X_calib, y_calib` (100 samples)
- All sets are **mutually exclusive**.
- `random_state` ensures reproducibility.
- Calibration set enables techniques like conformal prediction.
- Validation set helps with tuning and model selection.

Step 1: Load Raw CIFAR-10 Dataset & Metadata

- `tfds.load()`: Uses TensorFlow Datasets library to fetch the CIFAR-10 dataset.
- `batch_size=-1`: Loads the entire `train` and `test` splits in one batch.
- `as_supervised=True`: Returns data in `(features, label)` format.
- `with_info=True`: Also returns a metadata object containing dataset information.
- `label_names`: Extracted from metadata, lists the 10 class names (e.g., `'airplane'`, `'automobile'`, . . .).

Step 2: Convert to NumPy & Initial Train/Test Separation

- `tfds.as_numpy()`: Converts `tf.data.Dataset` objects into NumPy arrays.
  - Makes data easier to manipulate for standard ML workflows.
- CIFAR-10 from `tfds` is pre-split:
  - `dataset['train']`: Contains 50,000 training images.
  - `dataset['test']`: Contains 10,000 test images.
- The training data is named `X_train_full`, `y_train_full` to indicate it will be further split later.

Step 3: Create Final Train, Calibration, and Validation Sets

- The `train_valid_calib_split` function is used to split the initial training data.
- It takes `X_train_full`, `y_train_full` and divides them into:
  - `X_train`, `y_train`: Final training set.
  - `X_calib`, `y_calib`: Calibration set.
  - `X_val`, `y_val`: Validation set.
- The original test set `X_test`, `y_test` is kept separate and remains unchanged.

Step 4: Normalize Pixel Values

- CIFAR-10 images contain pixel values in the range $[0, 255]$.
- To normalize, divide each pixel value by 255 to scale it to the range $[0.0, 1.0]$.
- **Why normalization is important:**
  - Ensures all input features (pixels) are on a similar scale.
  - Helps neural networks converge faster during training.
  - Improves model performance and stability.

Step 5: One-Hot Encode Labels

- The original labels (`y_train`, `y_val`, etc.) are integers, e.g., 0 for `'airplane'`, 1 for `'automobile'`.
- `to_categorical()` is used to convert these integers into one-hot encoded vectors.
- **Example:**
  - If class `'airplane'` is label 0 (out of 10 classes), then:
  - `y_train_cat` for an airplane image becomes `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`.
- **Why?** One-hot encoding is required by many neural network loss functions, such as `categorical_crossentropy`, for multi-class classification.
- The original integer labels are retained alongside the one-hot encoded versions for flexibility.
- **Visual:**
  - `y_train = [0, 1, 2]` → `y_train_cat = [[1,0,0,...], [0,1,0,...], [0,0,1,...]]`

Step 6: Package and Return Datasets

- The processed data for each split is bundled into tuples.
- Each dataset tuple contains:
  - Normalized images (`X_...`)
  - Original integer labels (`y_...`)
  - One-hot encoded labels (`y_..._cat`)
- The function returns:
  - Four dataset tuples: train, calibration, validation, and test.
  - The list of class names: `label_names`.

Final Output Structure

- The function returns a complete, ready-to-use dataset for model development:
- **Returned Tuples:**
  - `train_set`: (`X_train_norm`, `y_train_int`, `y_train_onehot`)
  - `val_set`: (`X_val_norm`, `y_val_int`, `y_val_onehot`)
  - `calib_set`: (`X_calib_norm`, `y_calib_int`, `y_calib_onehot`)
  - `test_set`: (`X_test_norm`, `y_test_int`, `y_test_onehot`)
- `label_names`: A list of class names, e.g., `['airplane', 'automobile', ...]`.
- **Purpose:** Enables training, validation, calibration, and final evaluation on CIFAR-10 with preprocessed and well-structured data.

Model Architecture: Convolutional Layers

- `model = tfk.Sequential([...])`: Builds a linear stack of layers.
- **Conv Block 1:** Initial feature extraction
  - `Conv2D(16, (3,3), activation='relu', padding='same', input_shape=...)`
  - `MaxPooling2D((2,2))`: Reduces spatial dimensions.
- **Conv Block 2:** Learns deeper patterns
  - `Conv2D(32, (3,3), activation='relu', padding='same')`
  - `MaxPooling2D((2,2))`
- **Conv Block 3:** Captures high-level features
  - `Conv2D(64, (3,3), activation='relu', padding='same')`
  - `MaxPooling2D((2,2))`

Model Architecture: Dense Layers & Output

- `Flatten()`: Converts 3D feature maps to 1D vector.
- **Fully Connected Layers:**
  - `Dense(128, activation='relu')`
  - `Dense(64, activation='relu')`
  - `Dense(32, activation='relu')`
- **Output Layer:**
  - `Dense(10, activation='softmax')`
  - 10 classes (e.g., CIFAR-10); outputs probability scores that sum to 1.

Compiling and Returning the Model

- **Compiling the Model:** Prepares the model for training.
  - `loss`: Measures prediction error. Example: `CategoricalCrossentropy()` for multi-class classification with one-hot labels.
  - `optimizer`: Adjusts weights to minimize loss. Example: `Adam()`.
  - `metrics`: Tracks performance. Example: `['accuracy']` monitors classification success.
- **Returning the Model:**
  - The function returns the compiled `Sequential` model.
  - It is ready for training using `model.fit()`.
  - The architecture is reusable for image tasks like CIFAR-10; loss/optimizer/metrics can be customized.

TensorflowToMapie: Bridging TensorFlow/Keras with MAPIE

- **Class:** `TensorflowToMapie`
- **Purpose:** Makes a Keras Sequential model compatible with Scikit-learn-style tools like MAPIE.
- **Provides:** Scikit-learn API methods: `fit`, `predict`, `predict_proba`, and `__sklearn_is_fitted__`.
- Enables advanced post-hoc techniques like conformal prediction by integrating smoothly with MAPIE.

Fitting the Model (fit method)

- Accepts a Keras model and training/validation data.
- Internally uses Keras's `fit()` to train the model.
- Applies `EarlyStopping` to avoid overfitting:
  - Monitors validation loss.
  - Stops if no improvement (e.g., for 10 epochs).
  - Restores best weights after training.
- After training:
  - Stores the trained model.
  - Sets `trained_ = True`.
  - Infers class labels from output layer for Scikit-learn compatibility.

Prediction Scikit-learn Compatibility

- **predict_proba:** Returns probability scores from the trained model (e.g., softmax outputs).
- **predict:**
  - Uses predicted probabilities to find the most likely class.
  - Converts class indices into one-hot encoded format.
- **__sklearn_is_fitted__:**
  - Used by Scikit-learn utilities.
  - Returns `True` if model has been trained via `fit`.
- Ensures compatibility with MAPIE for uncertainty quantification.

MAPIE Setup & Prediction Set Generation

Goal: Generate conformal prediction sets using MAPIE methods on a pre-trained CIFAR-10 model.

1. Define Conformal Strategies

```
method_params = {
    "naive": ("naive", False),
    "lac": ("lac", False),
    "aps": ("aps", True),
    "random_aps": ("aps", "randomized"),
    "top_k": ("top_k", False)
}
```

Core Idea of Conformal Prediction

- **Goal:** Construct prediction sets $C(X_{\text{test}})$ such that the true label $Y_{\text{test}} \in C(X_{\text{test}})$ with high probability ($1 - \alpha$, e.g., 90%).
- **Steps:**
  1. Compute **nonconformity scores** to measure how "atypical" a label is.
  2. Use a **calibration set** to compute scores for known examples.
  3. Derive a threshold $\hat{q}$ from calibration scores (typically a quantile).
  4. Predict by including all labels with score $\leq \hat{q}$.

Naive Conformal Prediction

- **Intuition:** Include all classes whose model confidence (softmax) is high enough.
- **Nonconformity Score:** $s(x, y) = 1 - P(y|x)$
- **Calibration:**
  - Compute score for each calibration sample: $1 - P(y_i|x_i)$
  - $\hat{q}$: quantile of these scores
- **Prediction:**
  - For each class k, compute $1 - P(k|x_{\text{test}})$
  - Include class if score $\leq \hat{q}$
- **Setting:** `include_last_label=False`

LAC: Label-Agnostic Conformal

- **Intuition:** Similar to Naive; uses the same score and logic.
- **Nonconformity Score:** $s(x, y) = 1 - P(y|x)$
- **Calibration & Prediction:** Identical to Naive.
- **Why separate?** Difference is more theoretical; may differ in other tasks (e.g., regression).
- **Setting:** `include_last_label=False`

APS: Adaptive Prediction Sets

- **Intuition:** Build prediction set by accumulating probability mass until threshold is reached.
- **Nonconformity Score:** Sum of softmax scores up to true label's rank in sorted list.
- **Calibration:**
  - For each calibration sample, compute cumulative sum up to true label.
  - Derive $\hat{q}$ from these cumulative scores.
- **Prediction:**
  - Sort predicted probabilities.
  - Include top classes until cumulative score exceeds $\hat{q}$.
- **Setting:** `include_last_label=True`

Random APS: Randomized Adaptive Prediction Sets

- **Intuition:** Same as APS but refines decision boundary using randomization.
- **Randomization:**
  - For boundary class k, include with probability:

$$\frac{\tau - L}{P_k}$$

    where L is cumulative sum up to $k-1$, P_k is boundary class prob, and τ is threshold.
- **Benefit:** Closer empirical coverage to target $1 - \alpha$
- **Setting:** `include_last_label="randomized"`

Top-k Conformal Prediction

- **Intuition:** Fixed-size set: always pick the top k most probable classes.
- **Calibration:**
  - Rank true label in each calibration prediction.
  - $\hat{k}$: quantile of ranks across calibration set.
- **Prediction:**
  - Always include top $\hat{k}$ classes.
- **Limitation:** Less adaptive, constant-size prediction sets.
- **Setting:** `include_last_label=False`

Summary of Conformal Prediction Methods

Naive / LAC:

- Include all classes whose individual (softmax) scores exceed a global threshold.
- Simple and interpretable, but can lead to large prediction sets.

APS (Adaptive Prediction Sets):

- Construct sets by including top-ranked classes until a cumulative probability threshold is reached.
- More efficient with adaptive set sizes tailored to each sample's uncertainty.

Random APS:

- Enhances APS by adding randomization at the boundary of inclusion.
- Achieves coverage closer to the exact target level $1-\alpha$.

Top-k:

- Outputs a fixed-size prediction set of $\hat{k}$ classes.
- Simpler but less adaptive to individual prediction confidence.

MAPIE Setup (cont.)

2. Initialize Storage and Alpha Levels

We begin by preparing two things:

- Storage dictionaries to hold the predictions and prediction set scores for each method.
- A range of significance levels (α), which control the confidence level of our prediction sets. These typically range from 0.01 to 0.99 in steps of 0.01.

3. Loop Over Methods: Calibrate & Predict

For each conformal prediction method:

- Initialize a MAPIE classifier using the pre-trained model and specify the method's name and parameters.
- Calibrate the classifier using the calibration dataset.
- Generate prediction sets for the test data across all α levels, optionally including the true label depending on the method.

The predictions and their corresponding prediction set scores are stored for further evaluation.

Evaluation Metrics

Goal: Evaluate prediction sets using conformal prediction metrics.

Count Empty Prediction Sets
We define a metric to count how often the model returns an **empty prediction set**—i.e., cases where no label is assigned to an input. This helps identify how often the model is uncertain to the point of predicting nothing.

Initialize Metric Storage
We set up containers to record various evaluation metrics for each method, including:

- Number of empty prediction sets (nulls)
- Coverage (how often the true label is in the prediction set)
- Accuracy (how often the top prediction matches the true label)
- Size (average number of labels in the prediction sets)

Visualizing & Comparing Prediction Sets

Goal: Understand how different classification methods create prediction sets — not just predicting one label, but a set of likely options.

Why This Matters:

- Sometimes, giving a few likely labels is better than confidently choosing the wrong one.
- This helps us explore how uncertain the model is for each prediction.

What You'll See:

- A grid of images and their predicted label sets.
- **Each row** shows results from a different prediction method.
- **Each column** displays a different image.
- For each image-method pair, we list the predicted labels and their confidence.

Laying Out the Visuals & Showing Predictions

Step 1: Set Up the Grid

- A grid layout is created with rows for methods and columns for images.
- Each row is labeled with the name of the prediction method (e.g., APS, Top-k).

Step 2: Fill In the Grid

- Each cell in the grid shows one image.
- Alongside the image, we display the prediction set — the set of labels considered plausible.
- These labels are sorted by model confidence and selected based on the method's logic.

What This Tells Us:

- How many labels are included.
- Which labels the model thinks are most likely.
- Whether the true label is included.

Making Predictions Clear & Informative

Labeling the Predictions

- Each predicted label is shown with its name and confidence score.
- If the true label was not included in the prediction set, it's displayed with a warning.

Organizing the Display

- Text labels are neatly spaced so they don't overlap.
- Extra space is added when the true label is missing to highlight it clearly.

Color Coding

- **Green**: Correct label and included — ideal outcome.
- **Red**: Incorrect label but included — model is unsure.
- **Orange**: Correct label but missing — model made a critical error.

Overall Result: A side-by-side, visual comparison of how each method performs in terms of accuracy, uncertainty, and label set

Comparing How Different Prediction Methods Perform

Our Goal: Evaluate different ways of forming prediction sets using three key criteria:

- **Empty Sets:** How often does a method give no prediction at all? (Lower is better)
- **Coverage:** If we ask for 90% confidence, does the method actually include the correct label 90% of the time? (Closer to target is better)
- **Set Size:** How many labels are included when a method does predict? (Smaller is better — if coverage is good)

The Experiment:

- We vary the *desired confidence level* (called $1 - \alpha$).
- For each method, we track how the metrics change as we become more or less confident.

Comparing How Different Prediction Methods Perform

Visual Output:

- A row of three plots — one for each metric (Empty Sets, Coverage, Set Size).
- X-axis: Desired Confidence Level $(1 - \alpha)$.
- Each colored line: A different prediction method.

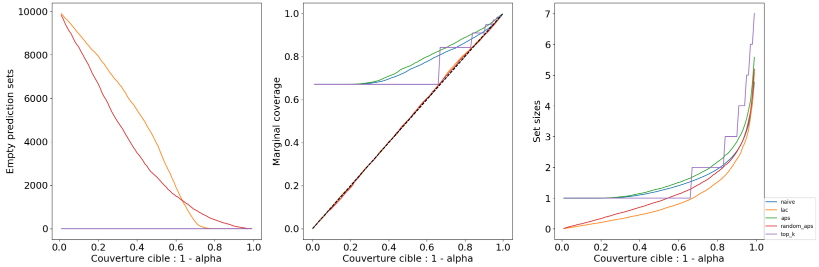


Figure:

Reading the Performance Graphs & Making Choices

How to Read the Plots:

- **X-axis:** Target Confidence $(1 - \alpha)$ — increases from left to right.
- **Y-axis:** Depends on the metric — Empty Sets, Coverage, or Set Size.
- **Lines:** Each colored line represents a different prediction method.

How to Interpret Each Plot:

- **Empty Sets:** Lower lines mean fewer images with no prediction — that's better.
- **Coverage:** Good methods follow the black diagonal line (perfect match between desired and actual coverage).
- **Set Sizes:** Smaller is better, but only if coverage is also high.

Reading the Performance Graphs & Making Choices

Making Trade-offs:

- No one-size-fits-all answer — some methods trade off size for better coverage.
- Best method depends on your goal: fewer misses, smaller predictions, or simpler results.

Analysis of Prediction-Set Size Distributions

Key Observations:

- **Naive & LAC:**
  - Peaks at size 1–2 (high precision).
  - Long tail up to 6–8 shows risk of undercoverage on hard samples.
  - LAC slightly more conservative—more singletons, fewer large sets.
- **APS:**
  - Peaks around size 2–3 (adaptive to uncertainty).
  - Tail extends to 9, reflecting larger sets when images are ambiguous.
- **Random APS:**
  - Smooth, gradual decline across sizes 1–8.
  - Randomization near the cutoff yields tighter average coverage.
- **Top-k:**
  - Single spike at size 3—always outputs exactly 3 labels.
  - No adaptivity: same set size for every image.

Analysis of Prediction-Set Size Distributions

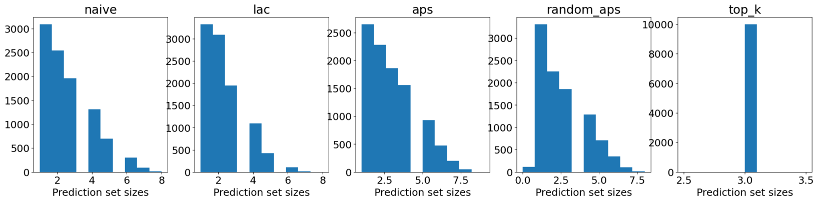


Figure:

Analysis of Prediction-Set Size Distributions

Trade-Offs to Consider:

- *Precision vs. Coverage:* Naive/LAC give small sets but may miss hard cases.
- *Adaptivity:* APS adjusts to image difficulty but can produce bulky sets.
- *Calibration:* Random APS balances set-size variability against more exact coverage.
- *Simplicity:* Top-k is predictable but can't flex to image uncertainty.

Conditional Coverage: The Fairness Question

The Problem: We ensure 90% global coverage, but does each class get treated equally?

Why It Matters:

- Overall coverage can mask failures on minority or hard classes.
- Systematic under-coverage reduces trust and fairness.

Next: How do we measure per-class reliability?

Measuring Conditional Coverage

Approach:

- For each method and each true class:
  - Select only samples of that class.
  - Compute the fraction whose true label was included.
- Plot these fractions as bars, with a dashed line at the 0.90 target.

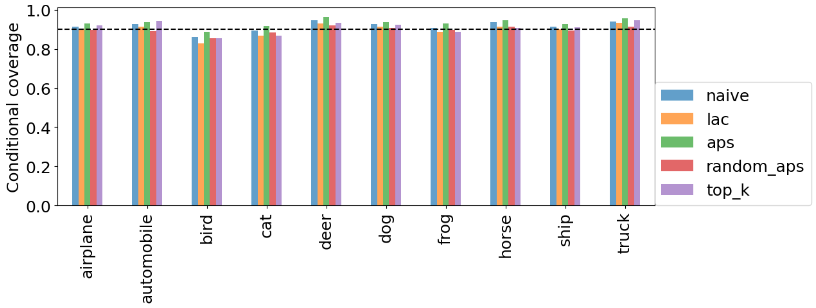


Figure:

Conditional Coverage: Key Insights

- "Deer," "Frog," "Horse," "Truck" mostly hit or exceed 90%.
- All methods dip on "Bird" $\rightarrow$ Bird images are systematically harder.
- Naive Top-k sometimes underperform on "Bird" and "Cat."
- LAC, APS, Random APS are more consistent across classes.

Implication: Investigate and potentially adapt methods for under-covered classes (e.g., "Bird").

Prediction-Set Heatmap: Understanding Confusions

The Problem: Which labels co-occur with the true label in a method's prediction sets?

Why It Matters:

- Identifies common class confusions (e.g., Cat Dog).
- Reveals semantic or visual similarities driving set membership.

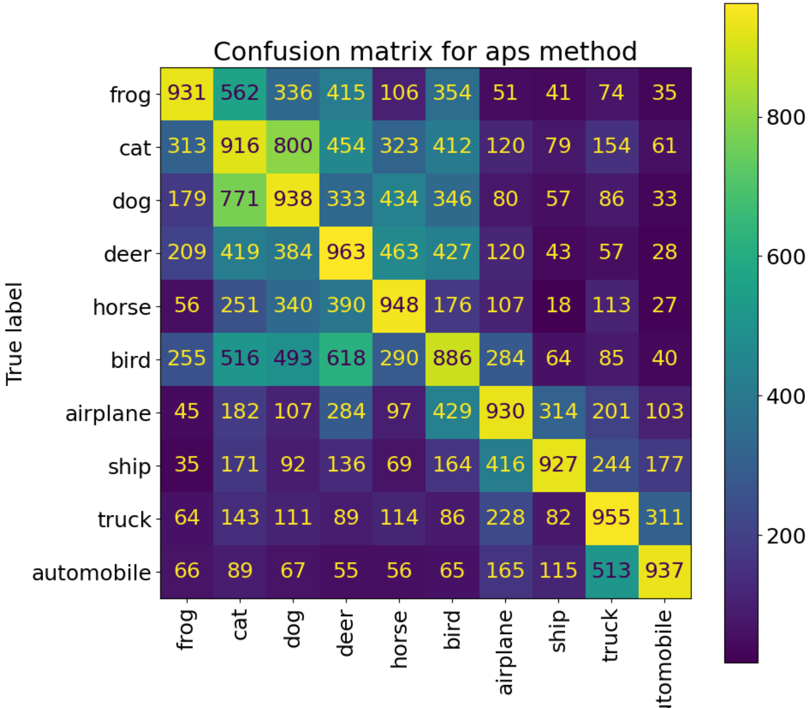
How It's Built:

- Rows = true labels; Columns = labels in the prediction set.
- Increment cell (i,j) for every sample of class i that includes j.
- Display as a heatmap of counts.

Heatmap Analysis: APS Method Example

- **Strong diagonal:** True labels are usually included (good).
- **Cat Dog:** Very high off-diagonal co-counts—classic confusion.
- **Bird Airplane:** Frequent pairing due to visual similarity.
- **Vehicles cluster:** Trucks, automobiles, ships, airplanes intermingle.
- **Frog:** Occasional confusion with cat/bird—dataset background effects?

[Visual: Heatmap titled "APS confusion matrix."]



True label	frog	cat	dog	deer	horse	bird	airplane	ship	truck	utomobile
frog	931	562	336	415	106	354	51	41	74	35
cat	313	916	800	454	323	412	120	79	154	61
dog	179	771	938	333	434	346	80	57	86	33
deer	209	419	384	963	463	427	120	43	57	28
horse	56	251	340	390	948	176	107	18	113	27
bird	255	516	493	618	290	886	284	64	85	40
airplane	45	182	107	284	97	429	930	314	201	103
ship	35	171	92	136	69	164	416	927	244	177
truck	64	143	111	89	114	86	228	82	955	311
automobile	66	89	67	55	56	65	165	115	513	937

Summary of Conditional Coverage Behavior

- **Naive:** Fails to guarantee both marginal and conditional coverage.
- **LAC:** Ensures marginal coverage, but shows the worst conditional coverage (largest per-class gaps).
- **Top-K:** Marginally valid with uniform set sizes, yet still under- or over-covers certain classes.
- **APS:** Achieves marginal validity with better conditional coverage via adaptive set sizes.
- **Random APS:** Adds randomness to APS, maintaining exact marginal coverage with slight conditional variability.

Takeaway: Progressing from *Naive* → *LAC/Top-K* → *APS* → *Random APS* improves control over coverage—marginal and conditional—at the cost of increased complexity or larger prediction sets.

Thank You!