Conformal Pipeline

02/27

Contents

0. Required Components

1. DGP and creating pseudo-outcomes
2. Data Splitting
3. Estimating beta.hat and beta.hat.ast
4. Prediction Step
5. Conformal Prediction

1. DGP, Creating Pseudo-Outcomes, and Individual-level Prediction

- **Step 1: Create pseudo-outcomes**
  - File: `generate_outcome_ast.rmd`
  - **Input:** Full respondent-level dataset containing `outcome_human` and covariates (`sex`, `race`, `age`, `marital_status`, `education`, `partyID`) `(line 27--28)`.
  - **Output:** Dataset with pseudo-outcome `outcome_ast (line 188)`.
  - **Procedure:**
    - Within each `study_id`, randomly sample 500 respondents without replacement to construct the estimation subset `(line 54--75)`.
    - For each survey question, fit a logistic regression model on the sampled subset:
      - Model specification: `outcome_human` $\sim$ `sex + race + age + marital_status + education + partyID (line 31, line 81)`.
      - Estimation method: standard logistic regression via `glm(..., family = binomial) (line 102--112)`.
    - Use the fitted model to compute predicted probabilities

      $$\hat{p}_i = \Pr(\texttt{outcome\_human}_i = 1 \mid X_i)$$

      for all respondents with complete covariates, including those in the estimation subset `(line 118--131)`.
      - For numerical stability, bound predicted probabilities to $[10^{-6}, 1 - 10^{-6}]$.
    - Generate the pseudo-outcome by simulation:

      $$\texttt{outcome\_ast}_i \sim \text{Bernoulli}(\hat{p}_i)$$

      using the bounded predicted probabilities `(line 169--177)`.

    - Save the fitted coefficient estimates from each logistic model. These coefficients define the groundtruth coefficients, i.e., β_d, that parameterize the DGP used to generate `outcome_ast (line 185)`.

- **Step 2: Create the pool of training, calibration and test observations**
  - File: `generate_outcome_ast.rmd`
  - **Input:** Full dataset with `outcome_ast` and covariates.
  - **Output:**
    - For the easier case: dataset with indicator column `is_train`.
    - For the time-extrapolation case: dataset with indicator columns `is_train`, `is_cal`, and `is_test`. `(line 228, 395)`
  - **Procedure:**
    - Easier case: for each study, randomly sample 500 respondents without replacement and assign `is_train=1`. All remaining observations have `is_train=0`. `(line 216--224)`
    - Time-extrapolation case: manually define the lists of studies in the training set and test set. Assign `is_train=1` and `is_test=1` respectively. Assign `is_cal=1` to all remaining observations not in the training or test sets. `(line 269--372)`
- **Step 3: Individual-level predictions (a.k.a. stabilization)**
  - File: `stabilize_xgboost_optuna.Rmd`
  - **Input:** Full dataset with `outcome_ast`, `outcome_silicon_original`, and `is_train`.
  - **Output:** Dataset with individual-level predictions `outcome_silicon_original_stabilized`.
  - **Procedure:**
    - Load embeddings `(line 103--152)`.
    - From observations with `is_train=1`, sample 3000 questions and 500 respondents without replacement to construct the training sample for stabilization `(line 158--203)`.
    - Impute missing AI prediction with 0 and 1 randomly with probability $p = 0.5$ `(line 196--197)`.
    - Assign a fold label to each question uniformly with $k = 5$ `(line 210--225)`.
      - Folds are split along questions to ensure that cross-validation is performed across questions rather than across respondents.

* Train an individual-level prediction model with 5-fold cross-validation:
  - Tuning metric: MAE with respect to $\hat{\mu}$ and $\hat{\beta}$ (`line 364--455`).
  - Model: XGBoost implemented by `xgboost`. Hyperparameters include `max_depth`, `min_child_weight`, `subsample`, `colsample_bytree`, `colsample_bylevel`, `gamma`, `lambda`, and `alpha`. Training settings include `n_trials=15`, `nrounds_max=500`, and `early_stopping_rounds=30` (`line 745--771`).
* Predict for all remaining observations not included in the stabilization training sample (`line 803--805`).
  - Generate binary predictions by random draw from $0, 1$ according to the predicted probability (`line 807--812`).
* Create hypothetically improved AI predictions:
  - Construct a map of wrong predictions by comparing `outcome_silicon_original` and `outcome_ast` (`line 915`).
  - Shuffle the indices of wrong predictions (`line 916`).
  - Select the first 10
  - For the selected subset, replace `outcome_silicon_original` with `outcome_ast` to generate hypothetically improved AI predictions.

---

2. Data Splitting

File: `evaluation_utils.R`

1. Easier Case Splitting (`make_cal_test_blocks_study_level`)

Used when `split_strategy = "random"`.

- **Step 1: Order respondents deterministically within study.**

  For each study s, respondents are sorted according to a deterministic pseudo-random key:

  $$k_{is} = \text{bitwXOR}(\text{rid\_int}_i, \text{seed}),$$

  where rid_int_i is the integer representation of `respondent_id`. This produces a reproducible random ordering without invoking `sample()` separately for each study.

- **Step 2: Partition ordered respondents into blocks.**

  Let N_s denote the number of non-train respondents in study s. The number of complete blocks equals $\lfloor N_s/B \rfloor$. Observations beyond position

  $$B\lfloor N_s/B \rfloor$$

  are discarded to preserve equal block sizes.

- **Step 3: Assign questions to blocks.**

  Let M_s denote the number of available questions in study s. The number of usable blocks is

  $$\min\left(\lfloor N_s/B \rfloor,\ M_s\right).$$

  Questions are sampled without replacement and matched one-to-one with blocks. Within each study, no two blocks share the same question.

- **Step 4: Allocate blocks to CAL and TEST.**

  Let $\phi =$ `FRAC_CAL` (default 0.85). A fraction ϕ of blocks is assigned to CAL and the remaining $1-\phi$ fraction to TEST. Sampling occurs at the block level:

  $$|\text{CAL}| \approx \phi \cdot \sum_s \min\left(\lfloor N_s/B \rfloor,\ M_s\right).$$

- **Step 5: Enforce caps on question counts.**

  If $|\text{CAL}| > Q_{\max}^{\text{cal}}$ (default 320), retain a uniformly random subset of $Q_{\max}^{\text{cal}}$ questions and their corresponding blocks; discard the remainder.

  If $Q_{\max}^{\text{test}}$ is specified and exceeded, apply the same truncation rule to TEST.

(`evaluation_utils.R`, lines 84–271)

2. Extrapolation Case Splitting (`make_cal_test_blocks_by_time`)

Used when `split_strategy = "time"`.

- **Step 1: Separate respondents by temporal labels.**

  Observations are pre-labelled:

  – `is_cal = TRUE` for earlier survey waves,

  – `is_test = TRUE` for later survey waves.

- **Step 2: Construct blocks within each temporal pool.**

  Apply the block construction procedure (Easier Case Splitting Steps 1–3) independently to:

  $$\{i : \texttt{is\_cal}_i = 1\} \longrightarrow \text{CAL blocks},$$

  $$\{i : \texttt{is\_test}_i = 1\} \longrightarrow \text{TEST blocks}.$$

CAL and TEST therefore contain disjoint questions and originate from temporally distinct survey waves, enabling time-extrapolation evaluation.

(`evaluation_utils.R`, lines 291–465)

2.3 Replication Seed Protocol

- **Step 1: Define replication-specific seed.**

  For replication r,

  $$\text{seed}_r = \texttt{BASE\_SEED} + r.$$

- **Step 2: Apply seed to all stochastic components.**

  The replication seed determines:

  - respondent hash-ordering,
  - question-to-block assignment,
  - CAL/TEST block allocation.

  Each replication therefore yields an independent yet reproducible partition.

(`run_experiment.R`, line 566)

---

3. Estimating Quantities of Interest

File: `dataset_transformer.R`

3.1 The Two Quantities

- **Step 1: Define per-question quantities.**

  For every question q in each of TRAIN, CAL, and TEST, estimate the pair $(\hat{\theta}_d, \hat{\theta}_d)$ by applying the same estimator twice:

  - once using human responses (`outcome_ast`),
  - once using LLM responses (`outcome_silicon_original_stabilized`).

- **Step 2: Compute quantity pairs.**

  The function `compute_beta_pairs()` (or `compute_mean_pairs()`) calls `dataset_transformer()` separately for human and LLM outcomes, then merges results by `question_id` into a wide-format table with one row per question.

3.2 Beta: Per-Question Logistic Regression

- **Step 1: Fit per-question logistic model.**

  For question d with n_d respondents, fit:

  $$\log \frac{P(Y_{id} = 1 \mid \mathbf{X}_i)}{1 - P(Y_{id} = 1 \mid \mathbf{X}_i)} = \alpha_d + \beta_d \cdot \texttt{partyID}_i + \gamma_d^\top \mathbf{Z}_i,$$

  where $\mathbf{Z}_i$ includes age, sex, education, race, and marital status.

  The quantity of interest is β_d, the log-odds ratio for `partyID`.

  Default estimator. In implementation, the model is fit using `arm::bayesglm()` (`method = "bayesglm"`) unless explicitly specified otherwise. The standard error $\widehat{\mathrm{SE}}(\hat{\beta}_d)$ is obtained from the model summary output.

- **Step 2: Estimation methods (alternatives).**

  Although `bayesglm` is the default and always used in the main pipeline, alternative estimators can be invoked by specifying `method`. These include:

  - *BayesGLM (`method = "bayesglm"`).* `arm::bayesglm()` with weakly informative Cauchy(0, 2.5) priors on all coefficients. The prior induces shrinkage toward zero, stabilising estimation under complete or near-complete separation. This is the standard estimator used in practice.
  - *GLM (`method = "glm"`).* `stats::glm()` with `family = binomial`. Accepted only if convergence is TRUE and all coefficients are finite.
  - *Firth (`method = "firth"`).* `logistf::logistf()` maximises the penalised likelihood:

    $$\ell_{\mathrm{F}}(\theta) = \ell(\theta) + \tfrac{1}{2} \log \det \mathbf{I}(\theta),$$

    where $\mathbf{I}(\theta)$ is the Fisher information matrix. Two-stage fitting is attempted:

    * First: `control(maxit = 100, maxstep = 5, maxhs = 10)`.
    * If needed: `control(maxit = 150, maxstep = 2, maxhs = 5)`.
  - *Ridge (`method = "ridge"`).* `glmnet::cv.glmnet()` with $\alpha = 0$:

    $$\hat{\theta}^{\mathrm{ridge}} = \arg\min_\theta \left[-\ell(\theta) + \lambda \|\theta\|_2^2 \right].$$

    λ is selected by 5-fold cross-validation or specified manually. Optional bootstrap SEs may be computed via B_{se} resamples.

- **Step 3: Construct design matrix within question.**

  For each question q:

1. Coerce Y_{iq} to $\{0, 1\}$ via `.coerce_y01()`.
2. Remove or impute rows with missing outcome or covariates (by default `na_strategy = "impute"`).
3. Build design matrix $\mathbf{X} = \text{model.matrix}(\sim ., \text{covariates})$.
4. Drop any constant column (zero variance).
5. Require $n_d \geq 10$ and at least two distinct values of Y_{iq}.

(`dataset_transformer.R`, lines ∼155–380)

3.3 Mean: Per-Question Proportion

- **Step 1: Compute per-question mean.**

  For question q with valid binary responses:

$$\hat{\mu}_d = \frac{1}{n_d} \sum_{i=1}^{n_d} Y_{id}.$$

- **Step 2: Compute standard error.**

$$\widehat{\text{SE}}(\hat{\mu}_d) = \sqrt{\frac{\hat{\mu}_d(1 - \hat{\mu}_d)}{n_d}}.$$

  If $\hat{\mu}_d \in \{0, 1\}$, then $\widehat{\text{SE}} = 0$.

3.4 Wide-Format Merge and Embedding Attachment

- **Step 1: Merge human and LLM quantities.**

  After computing $(\hat{\theta}_d)$ from `outcome_ast` and $(\hat{\theta}^*_d)$ from `outcome_silicon`, `compute_beta_pairs()` merges results by `question_id` into a wide-format table with one row per question.

- **Step 2: Attach question embedding if available.**

  If `question_embedding` is present, extract the first occurrence of each question's embedding $\mathbf{v}_d$ and attach it as a list-column, yielding one embedding vector per row.

(`dataset_transformer.R`, lines ∼590–650)

---

4. Question-level Prediction

Files: `trainer.R`, `training_metrics.R`

4.1 Predicting Quantities of Interests

- **Step 1: Define the training data.**

  Given the training set

  $$\mathcal{D}_{\text{train}} = \{(\hat{\theta}_d^*,\ \mathbf{v}_d,\ \hat{\theta}_d)\}_{d \in \mathcal{Q}_{\text{train}}},$$

  the goal is to learn a function f mapping $(\hat{\theta}_d^*, \mathbf{v}_d)$ to $\hat{\theta}_d$.

- **Step 2: Define the training objective.**

  The estimator $\hat{f}$ minimizes a loss $\mathcal{L}$ over training residuals:

  $$\hat{f} = \arg\min_f \ \mathcal{L}\Big(\{f(\hat{\theta}_d^*, \mathbf{v}_d) - \hat{\theta}_d\}_{q \in \mathcal{Q}_{\text{train}}}\Big).$$

- **Step 3: Apply the trained predictor to CAL and TEST.**

  The fitted model is evaluated on calibration and test questions to produce predictions $\hat{f}(\hat{\theta}_d^*, \mathbf{v}_d) \approx \hat{\theta}_d$.

4.2 Call Hierarchy

- **Step 1: Use a single public entry point.**

  All call sites route through `train_model()` in `training_metrics.R`.

- **Step 2: Dispatch by `cv_metric`.**

```
train_model()                                  [training_metrics.R]

    cv_metric   {"mae", "rmse"}  → .delegate() → trainer.R functions
    cv_metric = "q975"           → .train_xgboost_q975() or .train_rf_q975()
    cv_metric = "lp_N"           → .train_xgboost_lp()  or .train_rf_lp()
```

(`training_metrics.R`, lines 68–127)

4.3 Feature Matrix

`prepare_features()`

- **Step 1: Assemble the per-question feature vector.**

  For each question d, `prepare_features()` constructs:

  $$\mathbf{X}_d = \begin{cases} [\hat{\theta}_d^*] & \text{if } \texttt{use\_embeddings} = \text{FALSE} \quad (p = 1), \\ [\hat{\theta}_d^* \mid \mathbf{v}_d] & \text{if } \texttt{use\_embeddings} = \text{TRUE} \quad (p = 1 + K = 257). \end{cases}$$

- **Step 2: Normalize embedding storage formats.**

  When `use_embeddings = TRUE`, the embedding column may appear as a list-column, matrix, or array in R. `prepare_features()` standardizes these formats and outputs a numeric matrix suitable for downstream learners.

(`trainer.R`, lines 68–117)

4.4 Model Families

4.4.1 Linear / Ridge `train_linear`

- **Step 1: Handle the no-embedding case ($p = 1$).**

  When `use_embeddings = FALSE` (so $p = 1$), fit plain OLS via `lm()`:

  $$\hat{f}(\hat{\theta}^*_d) = \hat{a} + \hat{b}\,\hat{\theta}^*_d, \qquad (\hat{a}, \hat{b}) = \arg\min_{a,b} \sum_d (a + b\hat{\theta}^*_d - \hat{\theta}_d)^2.$$

  This avoids the `glmnet` degeneracy at $\lambda \to 0$ with a single predictor.

- **Step 2: Handle the embedding case ($p > 1$).**

  When `use_embeddings = TRUE` (so $p > 1$), fit ridge regression via `glmnet::cv.glmnet()` with $\alpha = 0$:

  $$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \sum_d (\mathbf{X}\beta - \hat{\theta}_d)^2 + \lambda \|\beta\|_2^2.$$

  λ is selected by cross-validation using `cv_metric` as the fold-level criterion.

(`trainer.R`, lines 168–220)

4.4.2 Lasso / Elastic Net `train_lasso`

- **Step 1: Use OLS when $p = 1$.**

  As in ridge, if $p = 1$ the implementation falls back to `lm()`.

- **Step 2: Use elastic net when $p > 1$.**

  For $p > 1$, fit `glmnet::cv.glmnet()` with mixing parameter $\alpha \in [0, 1]$:

  $$\hat{\beta}^{\text{en}} = \arg\min_{\beta} \sum_d (\mathbf{X}\beta - \hat{\theta}_d)^2 + \lambda\left[\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2\right].$$

  Default: $\alpha = 1$ (lasso). λ is selected by cross-validation.

(`trainer.R`, lines 484–539)

4.4.3 Random Forest `train_random_forest()`

- **Step 1: Specify the hyperparameter grid.**

  The tuning hyperparameter is `mtry` (the number of candidate features per split). The grid is:

  $$\mathcal{M} = \{1,\ \lfloor\sqrt{p}\rfloor,\ \lfloor p/3\rfloor,\ \lfloor p/2\rfloor,\ p\} \cap \mathbb{Z}_{>0}.$$

- **Step 2: Score candidates using OOB predictions.**

  For each $m \in \mathcal{M}$, fit a 200-tree forest and obtain OOB predictions $\widehat{\theta}_d^{\mathrm{OOB}}(m)$. The score depends on `cv_metric`:

  $$\mathrm{score}(m) = \begin{cases} \frac{1}{|\mathcal{Q}_{\mathrm{train}}|} \sum_d \left| \widehat{\theta}_d^{\mathrm{OOB}}(m) - \widehat{\theta}_d \right| & \text{if } \texttt{cv\_metric = "mae"}, \\ \texttt{fit\$prediction.error} \text{ (OOB MSE)} & \text{otherwise.} \end{cases}$$

- **Step 3: Fit the final forest with the best `mtry`.**

  Select $m^* = \arg\min_{m \in \mathcal{M}} \mathrm{score}(m)$ and fit the final 500-tree model using m^*.

- **Step 4: Guarantee inclusion of the primary feature.**

  When `use_embeddings = TRUE` ($p = 257$), the scalar $\widehat{\theta}_d^*$ may be excluded from candidate split sets when `mtry` $\ll p$. To prevent this, `always.split.variables = feature_col` forces $\widehat{\theta}_d^*$ into the candidate set at every node, regardless of `mtry`. When `always.split.variables` is active, `ranger` requires `mtry < p`, so p is removed from $\mathcal{M}$.

(`trainer.R`, lines 234–311)

4.4.4 XGBoost `train_xgboost()`

- **Step 1: Sample hyperparameters via random search.**

  Run random search over $T =$ `n_random_trials` configurations $\{\Omega_t\}_{t=1}^T$, where each configuration is sampled as:

  $$\eta_t \sim \mathrm{Uniform}(\log 0.05, \log 0.30), \quad d_t \sim \mathrm{Uniform}\{3, \ldots, 8\},$$
  $$\rho_t^{\mathrm{sub}} \sim \mathrm{Uniform}(0.5, 1.0), \quad \rho_t^{\mathrm{col}} \sim \mathrm{Uniform}(0.3, 1.0),$$
  $$w_t \sim \mathrm{Uniform}(1, 10), \quad \gamma_t \sim \mathrm{Uniform}(0, 2), \quad \lambda_t \sim \mathrm{Uniform}(0.1, 3), \quad \alpha_t \sim \mathrm{Uniform}(0, 1).$$

- **Step 2: Select boosting rounds via early stopping.**

  For each trial, `xgb.cv()` selects the number of rounds R_t^* via early stopping (patience = 30, maximum = 1000 rounds), by minimizing fold-averaged `cv_metric`. The best configuration is $\Omega^* = \arg\min_t \mathrm{cv\_score}_t$.

- **Step 3: Weight the primary feature $\widehat{\theta}_d^*$.**

  Let p be the total number of features ($p = 257$ with embeddings). Feature weights are set as:

  $$w_j = \begin{cases} p & \text{if feature } j \text{ corresponds to } \widehat{\theta}_d^*, \\ 1 & \text{otherwise.} \end{cases}$$

  This prevents $\widehat{\theta}_d^*$ from being dominated by embedding dimensions under column subsampling.

(`trainer.R`, lines 338–482)

4.5 Non-Standard CV Metrics

4.5.1 cv_metric = "q975" — 97.5th Percentile of |Residuals|

- **Step 1: Define the downstream motivation.**

  The prediction model is used inside a conformal prediction framework. The conformal interval half-width is proportional to:

  $$\hat{q} = Q^{\text{conf}}\Big(\{|\hat{\theta}_d^{\text{cal}} - \hat{f}(\hat{\theta}_d^{*\text{cal}})|\},\ \alpha\Big),$$

  the empirical $1-\alpha$ quantile of calibration residuals. Training to minimize the 97.5th percentile of training residuals directly targets downstream interval width.

- **Step 2: Define the HP-selection objective on OOF residuals.**

  The tuning criterion is:

  $$\mathcal{L}_{q_{0.975}}(\hat{f}) = Q_{0.975}\Big(\{|\hat{\theta}_d - \hat{f}(\hat{\theta}_d^*)|\}_{q\in\mathcal{Q}_{\text{OOF}}}\Big),$$

  computed over all out-of-fold (OOF) predictions jointly.

- **Step 3: Use a two-pass protocol for XGBoost.**

  - *Pass 1:* run `xgb.cv()` with a standard metric (`"mae"` for `reg:absoluteerror`, `"rmse"` otherwise) to determine R_t^* via early stopping.
  - *Pass 2:* run manual k-fold CV with fixed R_t^* (no early stopping) to collect OOF predictions $\hat{\theta}_d^{\text{OOF}}$, then compute:

    $$\text{score}_t = Q_{0.975}\Big(\{|\hat{\theta}_d - \hat{\theta}_d^{\text{OOF}}|\}_{q\in\mathcal{Q}_{\text{train}}},\ \text{type} = 8\Big).$$

- **Step 4: Special case for random forest.**

  For random forest, OOB predictions provide OOF residuals directly, so the second pass is not required.

4.5.2 cv_metric = "lp_N" — L_p Norm of Residuals

- **Step 1: Define the L_p scoring function.**

  For real $p \geq 1$,

  $$\mathcal{L}_p(\hat{f}) = \left(\frac{1}{|\mathcal{Q}_{\text{OOF}}|}\sum_{q\in\mathcal{Q}_{\text{OOF}}} |\hat{\theta}_d - \hat{f}(\hat{\theta}_d^*)|^p\right)^{1/p}.$$

  Special cases: $p = 1$ gives MAE, $p = 2$ gives RMSE, and $p \to \infty$ gives $\max|\hat{\theta}_d - \hat{f}|$.

- **Step 2: Motivation for $p > 2$.**

  For $p > 2$, $\mathcal{L}_p$ penalizes tail errors more than RMSE while using all residuals, giving a smoother, lower-variance alternative to quantile-based tuning.

- **Step 3: Implementation.**

  Implementation follows the same two-pass protocol as `cv_metric = "q975"`, replacing the quantile score with $\mathcal{L}_p$.

(`training_metrics.R`, lines 455–700)

4.6 Uniform Return Contract

- **Step 1: Return a named list.**

  All `train_*` functions return a named list with the following fields:

  - `$model`: fitted model object (e.g., `xgb.Booster`, `ranger`, `cv.glmnet`, `lm`).
  - `$predict`: `function(new_df) -> numeric vector`, ready-to-use.
  - `$feature_col`, `$embedding_col`, `$use_embeddings`: reproducibility metadata.
  - `$best_params`, `$best_nrounds`, `$best_cv_score`: XGBoost-only fields.

---

5. Conformal Prediction

File: `conformal_prediction.R`

5.1 Split Conformal Prediction: Framework

- **Step 1: Define calibration inputs.**

  Let $\mathcal{D}_{\text{cal}} = \{d_1, \ldots, d_{n_{\text{cal}}}\}$ be the calibration questions with pairs $(\hat{\theta}_{d_i}, \hat{f}_{d_i})$, where

  $$\hat{f}_{q_i} = \hat{f}(\hat{\theta}^*_{d_i}, \mathbf{v}_{d_i})$$

  is the question-level prediction model output for question d_i.

- **Step 2: Define conformity scores on CAL.**

  Let $s : (\hat{f}, \hat{\theta}) \mapsto \mathbb{R}_{\geq 0}$ be a conformity score measuring the deviation of $\hat{f}$ from $\hat{\theta}$. Define:

  $$s_i = s(\hat{f}_{q_i}, \hat{\theta}_{q_i}), \qquad i = 1, \ldots, n_{\text{cal}}.$$

- **Step 3: Compute the conformal quantile.**

  For miscoverage level α, compute

  $$k = \lceil (n_{\text{cal}} + 1)(1 - \alpha) \rceil, \qquad \hat{q}_\alpha = s_{(k)},$$

  where $s_{(k)}$ is the k-th order statistic of $\{s_1, \ldots, s_{n_{\text{cal}}}\}$.

- **Step 4: Form the finite-sample coverage statement.**

  For a new test question q_* drawn exchangeably with the calibration set,

  $$P\Big(s(\hat{f}_{d_*}, \hat{\theta}_{d_*}) \le \hat{q}_\alpha\Big) \ge 1 - \alpha.$$

  Equivalently, the prediction interval $C(q_*)$ induced by inverting the score satisfies $P(\hat{\theta}_{q_*} \in C(q_*)) \ge 1 - \alpha$.

- **Step 5: Use $(n_{\text{cal}} + 1)$ for exact coverage.**

  The $(n_{\text{cal}} + 1)$ term in the ceiling is required for exact $\ge 1 - \alpha$ coverage. Using n_{cal} instead yields coverage $\ge 1 - \alpha - 1/(n_{\text{cal}} + 1)$, which can be strictly below $1 - \alpha$.

  Example: for $n_{\text{cal}} = 320$ and $\alpha = 0.025$,

  $$k = \lceil 321 \times 0.975 \rceil = 313,$$

  so $\hat{q}_\alpha$ is the 313th/320 order statistic (97.81st percentile), slightly more conservative than the naive 97.5th percentile.

(`conformal_prediction.R`, `conformal_q()`, lines ∼35–60)

5.2 Method 1: Scaled Residual Score (SRS) — Default

- **Step 1: Define the SRS conformity score.**

  Define the scaled residual score:

  $$s_i = \frac{|\hat{f}_{d_i} - \hat{\theta}_{d_i}|}{\sigma_{d_i}},$$

  where σ_d is a question-specific scale parameter. Compared to raw residuals, SRS produces non-constant interval widths.

- **Step 2: Construct the SRS prediction interval.**

  For a new test question d_*,

  $$\mathcal{J}_{\text{SRS}}(d_*) = \left[\hat{f}_{d_*} - \hat{q}_\alpha \cdot \sigma_{d_*},\ \hat{f}_{d_*} + \hat{q}_\alpha \cdot \sigma_{d_*}\right].$$

  Larger σ_{d_*} yields wider intervals; smaller σ_{d_*} yields narrower intervals.

- **Step 3: Fit the scale model on data separate from CAL residuals.**

  The scale model is fit on a dataset disjoint from the calibration residuals used to compute $\hat{q}_\alpha$. The priority rule is:

  1. If TRAIN data is available: fit $\hat{\sigma}$ on $\{(\hat{f}_d,\ |\hat{f}_d - \hat{\theta}_d|)\}_{d \in \mathcal{D}_{\text{train}}}$, and use all of $\mathcal{D}_{\text{cal}}$ to compute $\hat{q}_\alpha$.
  2. Otherwise: split $\mathcal{D}_{\text{cal}}$ into two halves; fit $\hat{\sigma}$ on the first half and compute $\hat{q}_\alpha$ on the second half.

- **Step 4: Scale model for `predictor_type = "predictions"`.**

  Let $r_d = |\hat{f}_d - \hat{\theta}_d|$ be the absolute residual. The scale model regresses r_d on $\hat{f}_d$:

  - If $n < 250$: fit a linear model $\hat{\sigma}(\hat{f}_d) = a + b\hat{f}_d$.
  - If $n \geq 250$: fit a loess smoother with adaptive span

    $$\texttt{span} = \max\Big(0.3,\ \min\Big(0.75,\ \frac{300}{n}\Big)\Big).$$

  - Fallback chain: natural spline (`ns()`), then constant $\hat{\sigma} = \text{median}(r_d)$.

- **Step 5: Informativeness check for $\hat{\sigma}$.**

  Use the fitted scale model only if the predicted $\hat{\sigma}$ values are informative:

  $$\text{CV}(\hat{\sigma}) = \frac{\text{SD}(\hat{\sigma})}{\text{mean}(\hat{\sigma})} \geq 0.15 \quad \textbf{AND} \quad \text{SD}(\hat{\sigma}) \geq 0.02.$$

  Both conditions must hold simultaneously.

- **Step 6: Scale model for `predictor_type = "features"`.**

  Fit a scale model on the full p-dimensional feature vector $(\hat{\theta}^*_d, \mathbf{v}_d)$ using a model family (ranger/XGBoost/lasso/linear). The XGBoost variant uses `reg:absoluteerror`, targeting the conditional median of r_d.

- **Step 7: Clamp predicted scales.**

  Define:

  $$\sigma_{\text{floor}} = \max(Q_{0.05}(\{r_d\}),\ 0.05)\,, \qquad \sigma_{\text{cap}} = \max(Q_{0.95}(\{r_d\}),\ 3\,\sigma_{\text{floor}},\ 1.0)\,.$$

  Clamp predicted $\hat{\sigma}$ to $[\sigma_{\text{floor}}, \sigma_{\text{cap}}]$ to avoid degenerate narrow or wide intervals.

- **Step 8: Revert guard against overly wide intervals.**

  Compare the average SRS interval width at the smallest α to the residual method:

  $$\text{if} \quad \frac{1}{n_{\text{test}}} \sum_{d \in \mathcal{D}_{\text{test}}} 2\hat{q}_\alpha \sigma_d > \frac{1}{n_{\text{test}}} \sum_{d \in \mathcal{D}_{\text{test}}} 2\hat{q}^{\text{resid}}_\alpha,$$

then set $\hat{\sigma}_d \leftarrow \text{median}(r^{\text{cal}})$ for all q, reverting to a constant scale.

(`conformal_prediction.R`, `compute_srs()`, lines ∼100–650)

5.3 Method 2: Conformalized Quantile Regression (CQR)

- **Step 1: Fit quantile regression models on TRAIN.**

  Train quantile regression models at levels $\tau_{\text{lo}} = \alpha/2$ and $\tau_{\text{hi}} = 1 - \alpha/2$ using the training set. Let $\hat{q}_\tau(\mathbf{x}_d)$ denote the fitted quantile for question d at level τ.

- **Step 2: Define CQR conformity scores on CAL.**

  On calibration questions,

  $$s_i = \max\Big(\hat{q}_{\tau_{\text{lo}}}(\mathbf{x}_{d_i}) - \hat{\theta}_{d_i},\ \hat{\theta}_{q_i} - \hat{q}_{\tau_{\text{hi}}}(\mathbf{x}_{d_i})\Big)\,.$$

- **Step 3: Inflate quantile intervals using the conformal quantile.**

  Compute $\hat{q}_\alpha$ as in Section 5.1, and define for a new test question d_*:

  $$C_{\text{CQR}}(d_*) = \Big[\hat{q}_{\tau_{\text{lo}}}(\mathbf{x}_{d_*}) - \hat{q}_\alpha,\ \hat{q}_{\tau_{\text{hi}}}(\mathbf{x}_{d_*}) + \hat{q}_\alpha\Big]\,.$$

  CQR yields asymmetric intervals that adapt to the conditional distribution.

- **Step 4: Model options.**
  - `"linear"`: `quantreg::rq()` (fits multiple τ levels via a single LP).
  - `"xgboost"`: `xgboost` with `objective = "reg:quantileerror"` and `quantile_alpha =` (one model per τ).

- **Step 5: Pinball loss for XGBoost HP selection.**

  For quantile τ,

  $$\ell_\tau(y, \hat{\theta}) = (y - \hat{\theta})\big(\tau - \mathbf{1}\{y < \hat{\theta}\}\big)\,.$$

  Two tuning modes:
  - `cv_metric = "avg_pinball"`: minimize mean pinball loss across τ levels (default).
  - `cv_metric = "q975"`: minimize $Q_{0.975}(\{s_i\}_{i\in\text{OOF}})$ using the same two-pass protocol as Section 4.5.1.

(`conformal_prediction.R`, `compute_cqr()`, lines ∼670–950)

5.4 Method 3: Residual Score (Baseline)

- **Step 1: Define the residual conformity score.**

  $$s_i = |\hat{f}_{d_i} - \hat{\theta}_{d_i}|.$$

- **Step 2: Construct the residual prediction interval.**

$$C_{\text{resid}}(q_*) = \left[\hat{f}_{d_*} - \hat{q}_\alpha,\ \hat{f}_{d_*} + \hat{q}_\alpha\right].$$

This produces constant-width symmetric intervals and serves as the baseline.

(`conformal_prediction.R`, `compute_residual()`, lines ∼950–1020)

5.5 Multi-Alpha Support

- **Step 1: Accept a vector of miscoverage levels.**

  All three methods accept $\alpha = (\alpha_1, \ldots, \alpha_L)$.

- **Step 2: Return paired interval columns for each α_ℓ.**

  For each ℓ, return columns [`lower_`α_ℓ, `upper_`α_ℓ], with $\hat{q}_{\alpha_\ell}$ computed independently for each level.

- **Step 3: Maintain single-alpha backward compatibility.**

  For single-alpha experiments, legacy `lower` and `upper` columns are also added.

---

6. Evaluation

Function: `evaluate_intervals()` in `conformal_prediction.R` (lines ∼1030–1130)

6.1 Metrics

- **Step 1: Define test set and miscoverage levels.**

  For each miscoverage level α_ℓ and the test set $\mathcal{Q}_{\text{test}}$ of size n_{test}, evaluation is performed at the question level.

- **Step 2: Define achieved coverage.**

  Let $[L_d^\alpha, U_d^\alpha]$ denote the prediction interval for question d at level α. Achieved coverage is:

$$\hat{C}_\alpha = \frac{1}{n_{\text{test}}} \sum_{d \in \mathcal{D}_{\text{test}}} \mathbf{1}\left\{\hat{\theta}_d \in [L_d^\alpha, U_d^\alpha]\right\}.$$

- **Step 3: Define average interval width.**

$$\overline{W}_\alpha = \frac{1}{n_{\text{test}}} \sum_{d \in \mathcal{D}_{\text{test}}} (U_d^\alpha - L_d^\alpha).$$

- **Step 4: Report summary metrics and validity counts.**

Metric	Formula / Description
`achieved_coverage`	$\hat{C}_\alpha$
`coverage_gap`	$\hat{C}_\alpha - (1-\alpha)$; negative indicates under-coverage
`avg_width`	$\overline{W}_\alpha$
`median_width`	median$\{U_d^\alpha - L_d^\alpha\}$
`n_test`	Total number of test questions
`n_valid_truth`	Number of questions with non-NA $\hat{\theta}_d$
`n_valid_interval`	Number of questions with finite intervals satisfying $L_d^\alpha \leq U_d^\alpha$
`n_interval_bad`	Number of questions with NA, Inf, or $L_d^\alpha > U_d^\alpha$
`n_used`	Number of questions included in $\hat{C}_\alpha$ and $\overline{W}_\alpha$

- **Step 5: Handle invalid intervals conservatively.**

  If `treat_na_interval_as_miss = TRUE` (default), any NA or non-finite interval is treated as a **miss** (i.e., $\hat{\theta}_d \notin \mathcal{J}$). This is conservative and aligns with the theoretical coverage guarantee.

6.2 Outputs Written to Disk

- **Step 1: Write evaluation summary.**

  `evaluation_summary_rep_NNN.csv`: one row per α.

- **Step 2: Write question-level merged results.**

  `all_results_rep_NNN.csv`: merged CAL + TEST question-level data containing ($\hat{\theta}_d$, $\widehat{\text{SE}}$, $\hat{\theta}_d^*$, $\widehat{\text{SE}}$, $\hat{f}_d$, $L_d^{\alpha_\ell}$, $U_d^{\alpha_\ell}$) for all ℓ.
