A Details on CP and CP-inspired methods for classification

In the following table, $\ell = |Y|$, $T_A(n)$ is the training complexity on n examples of the nonconformity measure A, $P_A(m)$ is its running time for predicting m examples, n and m is respectively the number of training and test examples. In ICP and aggregated CP, $t \in \{1, ..., n\}$ is the size of the proper training set. In cross and aggregated CP, $K \in \{2, ..., n\}$ is a tuning parameter (number of folds).

	Train + Calibrate	Predict
CP	N/A	$\mathcal{O}((T_A(n) + P_A(1))n\ell m)$
ICP	$\mathcal{O}(T_A(t) + P_A(n-t))$	$\mathcal{O}((P_A(1) + n - t)\ell m)$
Cross CP	$\mathcal{O}((T_A((K-1/K)n) + P_A(n/K))K)$	$\mathcal{O}((P_A(1) + n/K)K\ell m)$
Aggregated CP	$\mathcal{O}((T_A(t) + P_A(n-t))K)$	$\mathcal{O}((P_A(1) + n - t)K\ell m)$

ICP. Algorithm 2 describes the algorithms for calibrating and computing a p-value with ICP. In the training phase, ICP trains the nonconformity measure A on the proper training set, $\{(x_1, y_1), ..., (x_t, y_t)\}$, and uses A to compute the nonconformity scores on the calibration set $\{(x_{t+1}, y_{t+1}), ..., (x_n, y_n)\}$ (Lines 1-6). Similarly to CP classification, an ICP classifier computes a p-value for every $\hat{y} \in Y$ by running `COMPUTE_PVALUE()` (Lines 8-11); differently from CP, the p-value is only based on the nonconformity scores computed during calibration and the one for the test object.

Algorithm 2 Inductive Conformal Prediction: computing a p-value for $(x, \hat{y})$

1 CALIBRATE$(x, y, \{(x_1, y_1), ..., (x_n, y_n)\}, t, A)$
2 $\quad Z_{train} = \{(x_1, y_1), ..., (x_t, y_t)\}$
3 $\quad Z_{calib} = \{(x_{t+1}, y_{t+1}), ..., (x_n, y_n)\}$
4 $\quad$ for i in t+1, ..., n
5 $\quad\quad \alpha_i = A((x_i, y_i); Z_{train})$
6 $\quad$ return $A(\cdot; Z_{train}), \{\alpha_{t+1}, ..., \alpha_n\}$
7
8 COMPUTE_PVALUE$(x, \hat{y}, A(\cdot; Z_{train}), \{\alpha_{t+1}, ..., \alpha_n\})$
9 $\quad \alpha = A((x, \hat{y}); Z_{train})$
10 $\quad p_{(x,\hat{y})} = \frac{\#\{i=t+1,...,n\,:\,\alpha_i \geq \alpha\}+1}{n-t+1}$
11 $\quad$ return $p_{(x,\hat{y})}$

CP alternatives. Algorithms of aggregated CP (Carlsson et al., 2014), cross CP (Vovk, 2015), CV+ and jackknife+ (Barber et al., 2019) can be found in the respective references. Note that CV+ and jackknife+, albeit originally designed for regression, can be extended to classification tasks (Appendix D in (Gupta et al., 2019)).

B Details on optimized nonconformity measures

B.1 LS-SVM

An LS-SVM model w is learned as a solution of:

$$w = \arg\min_{w \in \mathbb{R}^h} \rho||w||^2 + \sum_{i=1}^{n} (w^\top \phi(x_i) - y_i)^2 \,,$$

for regularization parameter ρ. The closed-form solution to this is:

$$w^* = \Phi[\Phi^\top \Phi + \rho I_n]^{-1} Y \,,$$

where $\Phi = [\phi(x_1), ..., \phi(x_n)]$, $Y = \{y_1, ..., y_n\}$, and I_n is the identity matrix of size $n \times n$.

The incremental&decremental learning method by Lee et al. (2019) requires storing an auxiliary matrix:

$$C = \Phi[\Phi^\top \Phi + \rho I_n]^{-1} \Phi^\top \,.$$

A description of how to learn incrementally or unlearn an example follows (Lee et al., 2019).

Incremental learning of one example. To learn (x_{n+1}, y_{n+1}), update the model as follows:

$$w_{new} = w + \frac{(C - I_q)\phi(x_{n+1})\left(\phi(x_{n+1})^\top w - y_{n+1}\right)}{\phi(x_{n+1})^\top \phi(x_{n+1}) + \rho - \phi(x_{n+1})^\top C \phi(x_{n+1})}$$
$$C_{new} = C + \frac{(C - I_q)\phi(x_{n+1})\phi(x_{n+1})^\top (C - I_q)}{\phi(x_{n+1})^\top \phi(x_{n+1}) + \rho - \phi(x_{n+1})^\top C \phi(x_{n+1})}\,.$$

Where q is the size of the kernel space.

Decremental learning of one example. To unlearn (x_i, y_i), update the model as follows:

$$w_{new} = w - \frac{(C - I_q)\phi(x_i)\left(\phi(x_i)^\top w - y_i\right)}{-\phi(x_i)^\top \phi(x_i) + \rho + \phi(x_i)^\top C \phi(x_i)}$$
$$C_{new} = C - \frac{(C - I_q)\phi(x_i)\phi(x_i)^\top (C - I_q)}{-\phi(x_i)^\top \phi(x_i) + \rho + \phi(x_i)^\top C \phi(x_i)}\,.$$

B.2 Bootstrap algorithm

Algorithm 3 shows the entire optimized bootstrap CP algorithm. In the training phase, B' bootstrap samples are generated, and for some of them (those that do not contain the placeholder "$*$") a classifier is trained. To compute a p-value for example $(x, \hat{y})$, the remaining classifiers are trained (after replacing "$*$" with $(x, \hat{y})$), and the predictions are computed as usual. A full implementation is provided in the code attached to this submission.

C Time complexity derivations

C.1 Simplified k-NN and k-NN

Standard. For simplicity, we only describe the complexity of Simplified k-NN; the complexity of k-NN is identical to the one derived for Simplified k-NN up to a linear factor.

Let us define a routine, `best_k(A)`, which returns the k smallest elements of a set A of size n. In our work, we instantiate this to Introselect, which runs in $\mathcal{O}(n)$ worst-case.[2]

The cost for computing the nonconformity measure $A((x, y); \{(x_1, y_1), ..., (x_n, y_n)\})$ for one example (x, y) requires computing the distances from x to the training points $\{(x_1, y_1), ..., (x_n, y_n)\}$, and selecting the k best. Overall, by using `best_k(A)`, this amounts to $\mathcal{O}(n)$. From the time complexity of CP classification (Section 2), we get that running CP with the (Simplified) k-NN nonconformity measure to predict m test examples takes $\mathcal{O}(n^2 \ell m)$.

Optimized. In the training phase, we precompute the distances and preliminary scores for the n training examples ($\mathcal{O}(n^2)$), and store both. To compute the nonconformity score A for one example (x_i, y_i) in the prediction phase, we only need to compute its distance from the test example (x, y), and update the provisional score α_i if this distance is smaller than one of the best k distances. This has cost $\mathcal{O}(1)$. The cost of CP classification with the optimized measure is therefore $\mathcal{O}(n\ell m)$.

C.2 KDE

Standard. Let P_K be the time to compute the kernel on 1 input. To compute the nonconformity score, we repeat this operation for all the training points ($\mathcal{O}(P_K n)$). Hence the cost of KDE CP classification is: $\mathcal{O}(P_K n \ell m)$.

Optimized. The training costs of this algorithm is $\mathcal{O}(P_K n^2)$. The cost for updating one nonconformity score with our optimization is $\mathcal{O}(P_K)$. Therefore, the cost of optimized KDE CP for classification is $\mathcal{O}(P_K n \ell m)$.

[2] In our implementation, we base `best_k` on `numpy`'s `argpartition`.

Algorithm 3 Optimized bootstrap CP algorithm.

TRAIN($\{(x_1, y_1), ..., (x_n, y_n)\}, B$)
    $Z^* = Z \cup \{*\}$ // "$*$" is a placeholder for the test example $(x, \hat{y})$
    $E_1, ..., E_n, E \leftarrow \{\}$

    // Associate at least B bootstrap samples to each training example and to "$*$"
    for B' in 1, 2, 3, ...
        $Z^*_{B'} \leftarrow$ sample $|Z^*|$ examples from Z^* with replacement
        for $i = 1, ..., n$
            if $(x_i, y_i) \notin Z^*_{B'}$
                Insert $Z^*_{B'}$ into E_i
        if $* \notin Z^*_{B'}$
            Insert $Z^*_{B'}$ into E
        if $|E| \geq B$ and $|E_i| \geq B$ for all $i = 1, ..., n$
            exit for loop

    // Pretraining for bootstrap samples that do not contain "$*$"
    for $i = 1, ..., n$
        for $Z_b \in E_i$
            if $* \notin Z_b$
                Train classifier g on Z_b, and replace element Z_b with $g(x)$ in E_i

    // Pretraining for placeholder "$*$". Note: by construction, no element of E contains "$*$"
    for Z_b in E
        Train classifier g on Z_b, and replace element Z_b with $g(x)$ in E

    return $E_1, ..., E_n, E$

COMPUTE_PVALUE($(x, \hat{y}), E_1, ..., E_n, E$)
    // Compute the nonconformity scores for the training examples
    for $i = 1, ..., n$
        $\alpha_i = 0$
        for $e \in E_i$
            if e is a pretrained classifier, call it $g(x)$
                $\alpha_i = \alpha_i - g^{\hat{y}}(x)$
            else (e is a bootstrap sample Z_b that contains "$*$")
                Replace "$*$" with $(x, \hat{y})$ in Z_b
                Train classifier g on Z_b
                $\alpha_i = \alpha_i - g^{\hat{y}}(x)$

    // Compute nonconformity score for the test example
    $\alpha = 0$
    for $g(x) \in E$
        $\alpha = \alpha - g^{\hat{y}}(x)$

    // Compute p-value
    $p_{(x,\hat{y})} = \frac{\#\{i=1,...,n\,:\,\alpha_i \geq \alpha\}+1}{n+1}$

    return $p_{(x,\hat{y})}$

C.3 LS-SVM

Standard. The running time of LS-SVM CP is dominated by training LS-SVM. Let $\mathcal{O}(n^\omega)$, $\omega \in [2,3]$ be this training cost. Then, CP classification is $\mathcal{O}(n^{\omega+1}\ell m)$.

Optimized. Training the optimized algorithm has the same cost as training standard LS-SVM, $\mathcal{O}(n^\omega)$. Let q be the dimensionality of the objects after feature mapping $\phi(x)$. To compute the nonconformity score for an example (x_i, y_i), we: i) update the model with the test example by using the method by Lee et al. (2019), ii) make a prediction for (x_i, y_i). The first operation has cost $\mathcal{O}(q^3)$), the second one requires n kernel evaluations and $\mathcal{O}(q)$ for the dot product. Therefore, predicting with optimized LS-SVM is $\mathcal{O}(q^3 n\ell m)$.

C.4 Bootstrap

Standard. Let $T_g(n)$ be the time needed to train the base classifier on n training points, and $P_g(m)$ its running time when computing a prediction for m points. To compute the bootstrap nonconformity measure once we need to train B base classifiers and run each one to compute a prediction. This amounts to $\mathcal{O}((T_g(n) + P_g(1))B)$. The overall complexity of bootstrap CP classification is $\mathcal{O}((T_g(n) + P_g(1))Bn\ell m)$.

Optimized. During the training phase, and for each training point, we will need to train (and make predictions for), in expectation, $BPr\left(* \notin Z_b\right)$, where $Pr\left(* \notin Z_b\right)$ is the probability that an example ("$*$", in this case) is not contained in a bootstrap sample of Z^* with replacement. It is easy to see that $BPr\left(* \notin Z_b\right) = B(1 - 1/n+1)^{n+1} \approx Be^{-1}$. If we repeat the argument for all training examples, the training phase (Lines 1-26 in Algorithm 3) takes time $(T_g(n) + P_g(1))Be^{-1}n$.

Computing the p-value for one point is obtained as the complement of the probability, $(T_g(n)+P_g(1))B(1-e^{-1})n$. This is a linear factor $(1-e^{-1}) \approx 0.632$ the speed of the original one. Overall, CP classification takes $(T_g(n)+P_g(1))B(1-e^{-1})n\ell m$ for classifying m test examples in ℓ labels.

Remark. The actual complexity of optimized bootstrap CP is generally lower than the one derived above. Indeed, in the above calculations we assumed that each bootstrap sample is used for just one point; however, some bootstrap samples (and respective classifiers) are in fact shared among several training points. Therefore, the effective number of classifiers one needs to train is only B', and not Bn. We show the relation between B and B' in Figure 5, which indicates that $B' < Bn$.

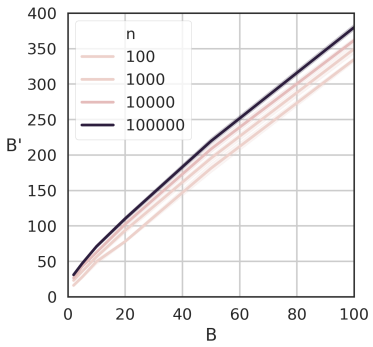

Figure 5. Relation between B, n, and B' for the optimized bootstrap CP algorithm.

C.5 IID test by Vovk et al. (2003)

Vovk et al. (2003) introduced an online algorithm for testing the exchangeability (or IID-ness) of a sequence of observations. At step n, having observed $\{x_1, ..., x_n\}$, it computes a p-value (Algorithm 1) for a new observation x_{n+1}. On the basis of the computed p-values, the test derives exchangeability martingales which can be used as the basis of an hypothesis test.

Suppose we use the k-NN nonconformity measure. Computing one p-value using standard k-NN CP is $\mathcal{O}(n^2)$. Since standard CP does not have any way of exploiting previous computations, the p-values have to be computed independently. The cost of processing n observations is $1^2 + 2^2 + ... + n^2 = \sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$. Hence, $\mathcal{O}(n^3)$.

When using the optimized k-NN CP, the cost of computing one p-value for x_{n+1} given n training examples is $\mathcal{O}(n)$; this includes the cost of training on the new observation. This means the cost of computing n p-values incrementally, as required by the IID test, is $1 + 2 + ... + n = \frac{n(n+1)}{2}$. That is, $\mathcal{O}(n^2)$.

D Memory costs

A standard CP implementation requires storing the entire training data Z, which has cost $\mathcal{O}(np)$, where n is the number of examples, p their dimensionality. In addition to this cost, the optimized CP versions we introduced have the following requirements.

(Simplified) k-NN. Store, for every training point: i) the largest among the best k distances, ii) the provisional score. This has cost $\mathcal{O}(n)$, negligible w.r.t. the $\mathcal{O}(np)$ already required by CP.

KDE. Store the n preliminary scores, $\mathcal{O}(n)$, which is negligible w.r.t. $\mathcal{O}(np)$.

LS-SVM. requires storing the model, $w \in \mathbb{R}^q$, and an auxiliary $q \times q$ matrix required by the method by Lee et al. (2019), where q is the dimensionality of the kernel space. Including the cost of standard CP, the memory used is $\mathcal{O}(np + q^2)$.

Bootstrap. In the optimized algorithm, we create B' bootstrap samples. Of them, on average $B'e^{-1}$ do not contain the placeholder $*$; we can therefore train them and compute a prediction for them (Lines 16-24). Storing these predictions has cost $\mathcal{O}(B'e^{-1})$. For the remaining ones, we need record the indices pointing to the augmented dataset Z^*, so that in the test phase we can reconstruct the bootstrap samples. We store n indices for each, totaling a memory cost of $\mathcal{O}(B'(1 - e^{-1})n)$. The overall cost of bootstrap CP is $\mathcal{O}(np + B'(1 - e^{-1})n)$. The relation between B', B, and n is shown in Figure 5.

E Experiment details

Hardware and multiprocessing. We conduct our experiments on a 2x Intel Xeon E5-2680 v3 (48 threads) machine, with 256 GB RAM. In these experiments, each time measurement is performed on a single core. We limit the number of cores available for the experiment, so as to prevent time measurements from being affected by other running processes (e.g., kernel tasks). We prevent `numpy` from automatically parallelizing matrix calculations.

Hyperparameters. We use the following hyperparameters for the nonconformity measures.

Method	Hyperparameters
Simplified k-NN & k-NN	Euclidean distance, $k = 15$.
KDE	Gaussian kernel. Bandwidth $h = 1$.
LS-SVM	Linear kernel, $\rho = 1$.
Bootstrapping	We instantiate bootstrapping to Random Forest, with $B = 10$ classifiers. Each classifier, a decision tree, is allowed to grow up to depth 10, and to select among $\sqrt{p}$ of features for a split, where p is the dimensionality of X.

When measuring time w.r.t. the training size n, we vary n in the space $[10, 10^5]$, by evenly separating 13 values on a log scale.[3]

F Simplified k-NN results

Due to lack of space, Figure 2 did not include results for Simplified k-NN. Figure 6 compares k-NN and Simplified k-NN, showing that they are very similar – indeed, their asymptotic time complexities are also identical.

G Experiments on `MNIST`

We conduct experiments on the `MNIST` dataset (LeCun et al., 2010), which includes 60k training examples and 10k test examples. This dataset's records have a much higher dimensionality than those considered in our previous experiments:

[3] Concretely, values for n are obtained with `numpy.logspace(1, 5, 13, dtype='int')`.

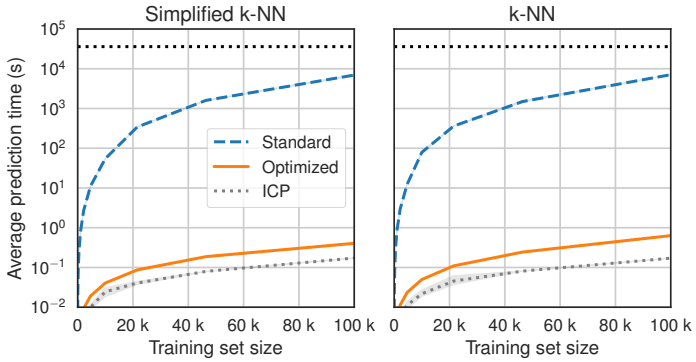

Figure 6. Comparison between standard and optimized k-NN and Simplified k-NN. ICP is used as a baseline. Prediction time for one test point w.r.t. the size of training data. Black dashed line is the experiment timeout (10 hours).

	CP	Optimized CP	ICP
NN	0s / T(1)	34m 5s / 7h 9m	22m 58s / 2h 38m
SimplifiedKNN	0s / T(1)	29s / 4h 36m	3m 47s / 1h 38m
KNN	0s / T(0)	34m 17s / 7h 21m	20m 25s / 2h 29m
KDE	0s / T(1)	1h 17m / 29h 13m	1h 30m / 6h 13m
RandomForest	0s / T(0)	48s / T(0)	21s / 1h 25m

Table 2. `MNIST` evaluation. Each entry reports *training/prediction* time for 60k training and 10k test points. $T(p)$ indicates that the timeout (48h) was reached before completing all the predictions, and that p predictions were made by then.

each object is a 28x28 pixel matrix (784 features in total). Furthermore, this is a 10-label classification setting; the number of labels strongly penalizes CP (and ICP), although this is an irreducible cost if we make no assumptions on Y (Section 8). (Fisch et al. (2021) recently made some developments w.r.t. this particular aspect.)

Time costs. We run standard and optimized CP on this dataset, by using the original training-test split. We do not include LS-SVM in this set of experiments, as it is specific to binary classification (although it could be extended by using e.g. a one-vs-all approach).

Table 2 indicates that the advantage of using optimized CP w.r.t. standard CP is substantial; in particular, for the fastest nonconformity measures, standard CP could make just 1 prediction (out of 10k test points) within the 48h timeout limit. Results also suggest that exact optimized CP is a practical alternative to ICP; for example, optimized Simplified k-NN CP run in 4.3 hours, while ICP with the same nonconformity measure run in 1.6 hours. Unfortunately, optimized Random Forest was unable to make any predictions within the 48h timeout; we hope this method can be further optimized in the future. Finally, we observe that optimized CP KDE was substantially worse than ICP; the reason is that, for the experiments on `MNIST`, we used arbitrary precision math to make KDE numerically stable – something that can be improved upon.

Statistical Efficiency of CP and ICP. As a byproduct of the above experiment, we are able to compare CP and ICP on the basis of their statistical power (efficiency). Note that an analysis on such a large dataset would not have been possible without using our CP optimizations.

We compare CP and ICP in terms of their *fuzziness* on the `MNIST` test set. The fuzziness of a set of p-values $\{p_{(x,y)}\}_{y\in Y}$, returned by CP or ICP as the prediction for test object x, is the average of the p-values excluding the largest one:

$$\sum_{y\in Y} p_{(x,y)} - \max_y p_{(x,y)} \,.$$

A smaller fuzziness indicates better performance (Vovk et al., 2016). We use a Welch one-sided statistical test for the null hypothesis H_0: "ICP has a smaller fuzziness (i.e., it is better) than CP". We reject the null hypothesis for a p-value < 0.01.

The table below indicates the fuzziness of the evaluated techniques for the `MNIST` dataset. An asterisk * indicates statistical significance. Random Forest CP was excluded, as it did not return predictions within the timeout (Table 2).

Results demonstrate CP is consistently better w.r.t. fuzziness than ICP. However, we observe that future work is needed to compare CP and ICP under various conditions (e.g., umbalanced data, distribution shift, ...). Our optimizations make this analysis feasible.

	CP	ICP
NN	**0.00047** $\pm$ **0.00105***	0.00065 $\pm$ 0.00143
Simplified k-NN	**0.04998** $\pm$ **0.07151***	0.05684 $\pm$ 0.07744
k-NN	**0.00066** $\pm$ **0.00125***	0.00098 $\pm$ 0.00174
KDE	**0.04494** $\pm$ **0.07005***	0.16791 $\pm$ 0.11729

H Does multiprocessing help?

We consider a multiprocessing implementation for CP classification. The CP implementation parallelizes Algorithm 1, which is run for every label and test point. Standard and optimized CP are parallelized in the same way. For the parallel versions, we employ a Python `Pool` of processes, as shown in the code included in the supplementary material.

We used the 48 threads machine described in Section 7, and made all the cores available for multiprocessing. We generated a dataset of size 1000, split it into training (%70) and test sets, and timed the sequential and parallel versions. Due to the high complexity of standard CP, we could not evaluate this for larger datasets. We report the measurements collected over 5 runs.

Results (Table 3) indicate that CP with standard nonconformity measures always benefits from parallelization, bringing at least one order of magnitude speed up. Conversely, optimized nonconformity measures give a mixed picture: except for LS-SVM and Random Forest, parallelization only brings mild improvements. Surprisingly, optimized k-NN CP is faster than the respective parallel version.

After this observation, we repeated the experiment just for optimized k-NN CP, for a dataset of 100k examples. In this case, parallelization indeed helps: prediction takes 1 hour for the parallel version, 3.5 hours for the sequential one. We conclude that the benefit of multiprocessing exists, but only for large datasets. We suspect this is due to the overhead of creating new processes, and that it can be further optimized in the future by working on the implementational details.

Table 3. Time comparison of sequential and parallel implementations, based on a dataset of 1000 examples with 30 features each. Time is measured in seconds.

		CP	CP Parallel
Standard	Simplified k-NN	74.60 $\pm$ 1.46	**3.95** $\pm$ **0.18**
	k-NN	82.14 $\pm$ 1.29	**4.11** $\pm$ **0.17**
	KDE	138.48 $\pm$ 1.49	**6.66** $\pm$ **0.28**
	LS-SVM	8852.18 $\pm$ 27.46	**624.74** $\pm$ **2.58**
	Random Forest	5061.67 $\pm$ 310.75	**225.75** $\pm$ **4.41**
Optimized	Simplified k-NN	1.29 $\pm$ 0.01	**0.43** $\pm$ **0.05**
	k-NN	**1.60** $\pm$ **0.27**	9.04 $\pm$ 0.33
	KDE	1.35 $\pm$ 0.14	**0.41** $\pm$ **0.00**
	LS-SVM	8.47 $\pm$ 0.58	**0.76** $\pm$ **0.01**
	Random Forest	2409.59 $\pm$ 546.21	**110.32** $\pm$ **32.50**