

RESEARCH ARTICLE

Evaluating tuning strategies for random forest hyperparameters with regards to prediction performance of clinical prediction models and computational time

J.A. Neve^{*1,2}

¹Department of Methodology and Statistics,
Utrecht University, Utrecht, Netherlands

²Julius Centre, University Medical Centre
Utrecht, Utrecht, Netherlands

Correspondence

*Corresponding author Email:
j.a.nevedemevergnies@students.uu.nl

Write my abstract here - 250 words max

KEYWORDS:

Write; my; keywords; here - six max

1 | INTRODUCTION

Machine learning models are increasingly used in medicine for diagnostic and prognostic purposes¹. Risk estimates are used to assist clinical decisions (e.g., whether to undergo treatment). A popular set of techniques in clinical research is tree-based methods², among which random forests are particularly prevalent^{3,4}. Random forests' results are influenced by hyperparameters (e.g., the number of predictors sampled to make a split)⁵, which need to be set before training a model. Previous studies have shown using optimal hyperparameters for a given dataset leads to little-to-no improvement in classification accuracy (i.e., the proportion of observations correctly classified) compared to the classification accuracy obtained when fitting the model with hyperparameter software defaults^{6,7}. Classification accuracy is however not sufficient to evaluate a model's clinical utility: identifying a patient as positive or negative may not carry sufficient information, as the patient's predicted risk is a key element of medical decision-making. Moreover, classification accuracy depends on the chosen classification threshold, i.e., the predicted risk value above which an observation is classified as positive. For clinical purposes, discrimination (i.e., positive patients have higher risk predictions than negative patients) and calibration (i.e., risk predictions reflect patients' true risk)⁸ are two much more important performance metrics. Models with low discrimination may misclassify patients, while miscalibrated models can lead to over- or undertreatment if over- or underestimating patients' risks. Some clinical studies have observed random forests to heavily suffer from miscalibration^{9,10}.

A well-known way to improve model performance is to perform hyperparameter tuning, which identifies hyperparameter values for which a performance metric (e.g., deviance) is optimal for a given prediction model. This procedure requires choosing which metric is optimised, which hyperparameters are tuned, and how candidate hyperparameter values are generated. The latter two can greatly impact computational time: generally, the more hyperparameters are tuned or the more candidate values there are, the larger computational time is. It is unclear whether this increased computational time is associated with model performance improvements. Hyperparameter search algorithms may consider all possible hyperparameter values and their combinations (e.g., grid search), or use information collected earlier in the tuning procedure to tune more efficiently (e.g., model-based optimisation)⁷. Computational time is important to consider in conjunction to the improvement in model performance: greater computational times and longer training periods of models may lead to a strain on resources and increase carbon emissions. For complex models, this can be as much as five times the average emissions of a car over its lifetime¹¹.

The optimal hyperparameter tuning strategy for balancing predictive performance and computational time is still an open question, particularly when considering clinically relevant performance metrics. Previous research suggests the most gain in

random forest performance as measured by classification accuracy, discrimination, and the Brier score could be made by tuning the number of predictors sampled to make a split and the proportion of the sample used to fit the tree¹². Yang and Shami¹³ found tuning using most hyperparameter search algorithms improved classification accuracy compared to default hyperparameters. Some search algorithms had greatly increased computational times without improving classification accuracy further than shorter search algorithms. So far, no study has compared different optimisation metrics' effects on model performance or computational time. Moreover, existing studies focusing on tuning procedures for random forests use high-dimensional datasets (datasets with more features than observations) whereas clinical research typically uses low-dimensional datasets (datasets with more observations than features), for which guidance regarding hyperparameter tuning is lacking¹⁴.

The current project aims to identify the optimal hyperparameter tuning strategies for balancing predictive performance and computational time of random forests in low-dimensional settings. There are three simulation studies in this project to identify (i) which hyperparameters to tune, (ii) which metric to optimise, and (iii) which search algorithm to use for optimal model performance. A post-hoc sensitivity analysis was conducted to examine (i) separation in calibration performances observed in certain hyperparameter tuning combinations and (ii) possible interactions between hyperparameter combinations and optimisation metrics.

2 | METHODS

We ran three simulation studies in order to identify optimal tuning strategies in controlled conditions. The studies were done sequentially to integrate conclusions from prior studies. We simulated data on which to compare tuning strategies. In this methods section, we detail the simulation methodology for each study following the ADEMP approach¹⁵.

2.1 | Aim

2.1.1 | Study 1: Hyperparameters to tune

Using datasets from the OpenML platform¹⁶, Probst and colleagues¹² found the number of predictors considered at a split and the sample fraction to be the two most influential hyperparameters on the discriminative performance of random forest prediction models. These findings only investigated the effect of tuning one or two hyperparameters at once. Our first study aimed to extend these findings by considering (i) model calibration in addition to model discrimination and (ii) larger combinations of hyperparameters, looking at the effect of tuning up to 5 hyperparameters at once. We examined the effect of tuning different combinations of hyperparameters on the performance of a prediction model and evaluated performance improvements in context of required computational times.

2.1.2 | Study 2: Optimisation metric

This study aimed to identify a metric to optimise in the tuning procedure leading to the best predictive performance of a random forest prediction model. We tuned the combination of hyperparameters considered optimal in Study 1.

2.1.3 | Study 3: Hyperparameter search algorithm

This study aimed to compare hyperparameter search algorithms' performances with respect to random forests' predictive performance and computational time. We tuned the combination of hyperparameters considered optimal in Study 1 using the optimisation metric selected in Study 2.

2.1.4 | Study 4: Post-hoc sensitivity analysis

This study aimed to examine how the selected hyperparameter values impacted calibration performance in random forests, and whether there was some interaction with optimisation metric. We focused on one scenario in which separation was observed in Study 1.

2.2 | Data-generating mechanism

All studies used the same data-generating mechanism. Different datasets were generated for each study.

2.2.1 | Data-generating scenarios

A full factorial simulation design was used to consider the influence of data characteristics on model predictive performance and computational time. The varying simulation factors were the number of candidate predictors p (range: 8, 16), the event fraction EF (range: 0.1, 0.3, 0.5), and the sample size n (range: $0.5N$, $1N$). N is the minimum sample size required to identify effects for a given number of regression coefficients (here, $1.25p$), expected event fraction, and expected AUC (here, 0.8), as calculated by Riley and colleagues¹⁷. A total of 12 (2^*3^*2) scenarios were considered. For each scenario, 500 training datasets were generated, yielding a total of 6,000 datasets per study. Scenario sample sizes are detailed in Table 1. Study 4 only considered one scenario, with 250 datasets.

TABLE 1 Data-generating scenarios

<i>p</i>	<i>EF</i>	<i>n</i>
8	0.10	381
16	0.10	762
8	0.30	172
16	0.30	344
8	0.50	193
16	0.50	293
8	0.10	762
16	0.10	1523
8	0.30	344
16	0.30	688
8	0.50	385
16	0.50	585

Training and validation datasets were simulated under a logistic model with strong interactions. For each observation i ($i = 1, \dots, N$), predictors \mathbf{x}_i were drawn from a p -variate normal distribution with mean $\boldsymbol{\mu} = \mathbf{0}$, each predictor's variance set to 1, and two-way predictor correlations set to 0.2. We included $0.25p$ two-way interactions, with the h^{th} ($h = 1, \dots, 0.25p$) interaction being the product of the h^{th} and the $(h + p/4)^{th}$ predictors. The binary outcome y_i was drawn from a Bernoulli distribution conditional on the predictor values (Formula 1).

$$P(y_i = 1|\mathbf{x}_i) = \frac{1}{1 + \exp(-(\beta_0 + \sum_{j=1}^{p/2} \beta_j x_{ij} + \sum_{k=1+p/2}^{3p/4} 2\beta_k x_{ik} + \sum_{h=1}^{0.25*p} \gamma_{ih} x_{i(h+0.25p)}))}. \quad (1)$$

In Formula 1, β_0 , β , γ are respectively the intercept, main effect regression coefficient, and interaction effect regression coefficient of the data generating model, hereafter called “true effects”.

A validation dataset ($n = 100,000$ for study 1 and $n = 10,000$ for studies 2 to 4) to evaluate model performance was generated for each training dataset. The size of the validation dataset was reduced after study 1 due to storage constraints.

2.2.2 | True effect estimation

True effects were determined for each unique combination of p and EF . Let k denote a given combination. For each combination, the intercept $\beta_0^{(k)}$, predictor main effects $\boldsymbol{\beta}^{(k)}$, and predictor interaction effects $\boldsymbol{\gamma}^{(k)}$ were estimated using a large sample approximation ($n = 100,000$). The first $p/2$ main effects were set to be equal (i.e., $\beta_1^{(k)} = \beta_2^{(k)} = \dots = \beta_{p/2}^{(k)}$). The next $p/4$ main effects were set to be twice the strength of the first $p/2$ main effects (i.e., $\beta_{p/2+1}^{(k)} = \beta_{p/2+2}^{(k)} = \dots = \beta_{3p/4}^{(k)} = 2\beta_1^{(k)}$). The final $p/4$ main effects were set to 0. All interaction effects $\boldsymbol{\gamma}^{(k)}$ were set to be equal (i.e., $\gamma_1^{(k)} = \gamma_2^{(k)} = \dots = \gamma_{0.25p}^{(k)}$). The R function `optim` was used to minimise a loss function measuring the sum of the squared difference between (i) 0.7 and the observed AUC in a model with no interactions, (ii) 0.8 and the observed AUC in a model with interactions, and (iii) the targeted event fraction and the average estimated probability $P(y_i = 1|\mathbf{x}_i)$ in the simulated dataset. This was repeated 20 times for each scenario. Each generated set of true effects was used to generate a dataset of $n = 100,000$, on which the AUC (with and without interaction) and

TABLE 2 Performance of selected true effects

p	EF (target)	β_0	β	γ	AUC (no interaction)	AUC (interaction)	EF (obtained)
8	0.10	-2.98	0.14	0.66	0.71	0.80	0.10
8	0.30	-0.94	0.24	-0.84	0.70	0.80	0.30
8	0.50	-0.22	0.22	0.83	0.70	0.80	0.50
16	0.10	-3.06	0.07	0.44	0.70	0.80	0.10
16	0.30	-0.85	0.14	-0.55	0.70	0.80	0.30
16	0.50	-0.31	-0.13	0.55	0.70	0.80	0.50

prevalence were measured. For each scenario, the true effect was selected to be the set of effects with the smallest sum of absolute differences between (i) target prevalence and observed prevalence, (ii) 0.7 and the AUC of a model with no interactions, and (iii) 0.8 and the AUC of a model with interactions. True effects' performances were assessed by generating a new dataset and measuring AUC and prevalence (Table 2).

2.3 | Simulation methods

2.3.1 | Study 1: Hyperparameters to tune

We varied which hyperparameters were tuned when fitting a random forest model using the R package `ranger`¹⁸ via the R package `caret`¹⁹. We used grid search to optimise deviance. 5-fold cross-validation was used.

Random forests can have up to ten hyperparameters¹⁸, some of which are only used if another hyperparameter has a certain value (e.g., `num.random.splits` is an additional hyperparameter when `splitrule` is set to `extratrees`). Considering all possible combinations of all these hyperparameters leads to 1023 possible combinations. Due to the unfeasibility of such a large-scale approach, we only studied hyperparameters included in a previous study by Probst and colleagues, which focused on hyperparameters' effects on predictive performance¹² in order to expand the findings by considering hyperparameters' combined effects. This led to the inclusion of `mtry` (the number of predictors randomly sampled to make a split), `min.node.size` (the minimum number of observations for a node to be formed, i.e., the point at which a split should not be computed regardless of impurity), `replace` (whether the data used to fit a single tree is sampled with or without replacement), and `sample.fraction` (the proportion of the data used to fit a single tree). Although `num.trees` (the number of trees used in the random forest) was also included in the prior study, it was not included in this study as it has been shown that this hyperparameter should simply be set to a high, computationally feasible, number²⁰. Additionally, we included `splitrule` (the way in which a split is picked) as it can be tuned by commonly used software such as `caret`¹⁹ (which does not tune other hyperparameters such as `replace`). Hyperparameter default values and tuning ranges used in this study are presented in Table 3.

Considering all possible combinations of these five hyperparameters still leads to an unfeasible number of 31 combinations. We used Probst and colleagues'¹² results to further reduce this number. They found `mtry` and `sample.fraction` was the hyperparameter combination that led to the most gain, and that `sample.fraction` and `min.node.size` had a similar effect. The commonly used tuning package `caret`¹⁹ allows for the tuning of `min.node.size`, but not `sample.fraction`. Thus, we only considered hyperparameter combinations that included at least `mtry` and `min.node.size`, as this combination could be expected to perform well.

This yielded 8 hyperparameter combinations. Hyperparameters not included in a given combination were set to their default value. A model using the default hyperparameters was fit to establish the baseline. All considered combinations were used to fit a random forest on each training dataset.

2.3.2 | Study 2: Optimisation metric

We varied the metric to optimise when fitting a random forest using the R package `ranger`¹⁸ via the R package `caret`¹⁹. Each dataset was tuned once using each candidate optimisation metric. We used grid search to tune the hyperparameters considered optimal in Study 1. Tuning ranges were identical to those of study 1 (Table 3). 5-fold cross-validation was used as part of the tuning procedure. The considered candidate optimisation metrics were the deviance (Formula 2; measuring the total deviations between predicted risk and the true outcome), the logarithmic loss (Formula 3; measuring the mean deviations between predicted risk and the true outcome, thus), the AUC (a measure of how well the classes are separated), the calibration intercept

TABLE 3 Hyperparameter tuning ranges in studies 1 and 2

Hyperparameter	Default	Range
mtry	\sqrt{p} (rounded down)	1-p
min.node.size	1	1-10
replace	TRUE	TRUE, FALSE
sample.fraction	1	0.1, 0.2, ..., 0.9, 1
splitrule	gini	gini, hellinger, extratrees*

*For splitrule = extratrees, an additional parameter should be considered regarding the number of random splits to consider. This was set to its default of 1.

TABLE 4 Optimisation metric targets

Metric	Target	Range of possible values
Deviance	0	[0, ∞]
Logarithmic loss	0	[0, 1]
AUC	1	[0.5, 1]
Brier score	0	[0, 1]
Calibration intercept*	0	$[-\infty, \infty]$
Calibration slope**	1	[0, ∞]
Classification accuracy	1	[0, 1]
Cohen's Kappa	1	[0, 1]

*optimised by minimising the squared value.

**optimised by minimising the squared natural log of the value.

(the difference between the average predicted risk and the true event rate), the calibration slope (the extent of over- or under-estimation of risk), the Brier score (Formula 4; this can be seen as a composite measure of calibration and discrimination²¹), the classification accuracy (the proportion of correctly classified observations) and Cohen's Kappa (the proportion of correctly classified observations, adjusted for chance). These are summarised in Table 4 alongside their target values.

$$\text{Deviance} = -2 * \sum_{i=1}^n (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)). \quad (2)$$

$$\text{Logarithmic loss} = \frac{1}{n} \sum_{i=1}^n (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)). \quad (3)$$

$$\text{Brier score} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2. \quad (4)$$

2.3.3 | Study 3: Hyperparameter search algorithm

We varied the hyperparameter search algorithm used when fitting a random forest. We tuned the hyperparameters selected from Study 1 and optimised the metric selected from Study 2. Grid search, random search (both using the R package caret¹⁹), and model-based optimisation (using the R package tuneRanger⁷) were used.

Grid search takes a set of possible values for each hyperparameter and uses every possible combination to fit a random forest. Random search computes a given number of random combinations of hyperparameter values and uses each of these combinations to fit a random forest. For both of these search algorithms, the combination of hyperparameter values leading to the best performance (here, as measured by the deviance) is selected to be the optimal hyperparameter values. These two algorithms were performed using 5-fold cross-validation. Grid search used a grid with mtry going from 1 through to p and min.node.size going from 1 through to 10. Random search was done by computing a grid with mtry going from 1 through

to p and `min.node.size` going from 1 through to $EF * N$, then randomly selected a number of rows equal to $p * 10/2$ (such that the number of iterations was half the number of iterations in grid search).

Model-based optimisation first computes a given number (here, 30) of random combinations of hyperparameter values and uses each of these combinations to fit a random forest and assess its performance (here, using the deviance). Then, it fits a regression model with the hyperparameter values as predictors and the performance as the outcome to identify a combination of hyperparameter that is expected to perform better. This combination is tried, and the regression model is fit again including the new point. This is repeated a given number of times (here, 70). The best hyperparameter combination is then assessed to be the mean of the 5% best-performing combinations. This algorithm uses out-of-bag performance rather than cross-validation. We refer to Probst and colleagues⁷ for details.

2.3.4 | Study 4: Post-hoc sensitivity analysis

We focused on the scenario where $p = 16$, $EF = 0.5$, $n = 1N$. We considered all combinations of hyperparameters included in Study 1. Optimisation metrics that were compared were AUC and deviance. Deviance and logarithmic loss are two mathematically extremely similar measures; deviance was chosen in this case to increase comparability to Study 1.

2.4 | Performance measures

The same outcomes were used in all studies. Primary outcomes were defined as those which were used to advise on the best tuning procedures. These measures were AUC, calibration slope, root mean square of the log of the calibration slope over all the runs of a scenario ($\text{RMSD}(\text{slope})$), and computational time. Discrimination was measured by the AUC and assessed whether positive observations have higher predicted risk than negative observation. Calibration was measured by the calibration slope and assessed the extent to which predicted risks reflect true risks. $\text{RMSD}(\text{slope})$ assessed the extent to which the calibration slopes varied within a data-generating mechanism and how much they differed from the ideal value of 1, as used by Van Calster and colleagues²². Computational time was the amount of time for which a given tuning procedure ran, measured in seconds.

Candidate optimisation metrics from Study 2 were also included as secondary outcomes in all studies. These were the classification accuracy, the calibration intercept, the Brier score, the logarithmic loss, and Cohen's Kappa.

Model performance metrics were estimated using the predictions of the model on a validation set independently generated under the same data generating mechanisms. For each tuning procedure, we computed the average and Monte Carlo error of each of these performance measures. We visualized whether certain hyperparameter combinations, optimisation metrics, or hyperparameter search algorithms had a notably larger runtime compared to others, without showing a substantial increase in performance. The best hyperparameter combination, optimisation metric, and hyperparameter search algorithm were assessed considering all primary outcomes.

2.5 | Software

Data was simulated and tuning procedures were performed on a high-performance computer. This was done in R version 4.2.2, using packages `MASS`, `dplyr`, `ranger`, `PROC`, `psych`, and in the case of study 3, `tuneRanger` [INCLUDE CITATIONS FOR ALL OF THESE]. Summary statistics and figures were generated on a personal computer using R version 4.2.3²³ using packages `tidyverse`, `viridis`, and `xtable` [CITE ALL OF THESE].

2.6 | Reproducibility

All code used to generate and analyse data is available at [LINK TO THE GITHUB REPO].

Data cannot be perfectly reproduced due to the nature of this study: the runtime variable is highly dependent on the system on which the script is run and other tasks being performed, so it can never be perfectly reproduced. Additionally, the high-performance computer had two operating systems responding differently to seeds [INCLUDE NAMES]. Runs were assigned to operating systems at random. However, these issues are only a concern with regard to an exact reproduction of the data, and should only lead to minor differences in results and no differences in interpretation.

TABLE 5 Average performance of hyperparameter combinations (aggregated over all scenarios). Rows are sorted in ascending order of runtime.

Tuned hyperparameters	AUC	Calibration slope	RMSD(slope)	Runtime (seconds)
	Mean (SD)	Median (IQR)		Mean (SD)
None	0.7 (0.02)	0.89 (0.26)	0.23	1.8 (0001.3)
mtry + min.node.size	0.7 (0.02)	1.07 (0.28)	0.21	267.9 (0298.2)
mtry + min.node.size + replace	0.7 (0.02)	1.05 (0.28)	0.21	614.7 (0691.8)
mtry + min.node.size + splitrule	0.71 (0.02)	1.24 (0.44)	0.36	746.0 (0739.0)
mtry + min.node.size + sample.fraction	0.7 (0.02)	1.16 (0.38)	0.30	1712.9 (1794.2)
mtry + min.node.size + replace + splitrule	0.71 (0.02)	1.2 (0.46)	0.36	1734.4 (1739.6)
mtry + min.node.size + sample.fraction + replace	0.7 (0.02)	1.14 (0.4)	0.31	3776.2 (4009.5)
mtry + min.node.size + sample.fraction + splitrule	0.71 (0.02)	1.24 (0.49)	0.47	4800.4 (4493.4)
mtry + min.node.size + sample.fraction + replace + splitrule	0.71 (0.02)	1.21 (0.49)	0.46	10668.1 (10140.9)

3 | RESULTS

3.1 | Study 1: Hyperparameters to tune

The average and Monte Carlo error of primary outcomes for each hyperparameter combination are presented in Table 5. Effects were homogeneous across scenarios. We illustrate results specific to the scenario where $p = 8$, $EF = 0.3$, and $n = 1N$ in Figure 1. Calibration plots for the scenarios where $EF = 0.5$ are presented in Figure 2. Results for other scenarios can be visualised at [THIS WILL BE A LINK TO A SHINY APP].

Large increases in runtime were observed when tuning 4 or more hyperparameters. Tuning large combinations of hyperparameters also led to noticeably worse calibration performance, but no observable change in other performance measures (Figure 1, Table 5). Varying hyperparameter tuning combinations had very limited effects on discrimination, with a mean AUC of 0.71 ($SD = 0.02$) across all hyperparameter combinations. A large effect was observed on model calibration and computational time. Model calibration varied considerably within (Figure 1) and between (Table 5, Figure 1, Figure 2) combinations. The best calibration performance was obtained by tuning `mtry` and `min.node.size`, which had a median calibration slope of 1.07 across all scenarios. This performance was among the most stable with an IQR of 0.28. Tuning `mtry`, `min.node.size`, and `replace` performed similarly well, but took twice as long to run. The worst calibration performances were observed when tuning `mtry`, `min.node.size`, `sample.fraction`, and `splitrue`, or these four hyperparameters and `replace`. Including `splitrue` in any combination led to a median calibration slope above 1.21 and an RMSD(slope) above 0.36, while combinations without `splitrue` have median calibration slopes below 1.15 and RMSD(slope) below 0.32. These observations are also visible in the calibration plots: calibration seems best when doing minimal tuning [OR SOMETHING TO THIS EXTENT ONCE I HAVE SAID PLOTS]. `mtry` and `min.node.size` was thus selected as the best combination to tune.

Secondary outcomes showed similar patterns (Figure 1). Classification accuracy, Brier score, and logarithmic loss showed very little variation both within and between hyperparameter combinations. Calibration intercept and Cohen's Kappa showed variations within, but not between, hyperparameter combinations.

3.2 | Study 2: Optimisation metric

Based on the findings of Study 1, we tuned `mtry` and `min.node.size`. The average and Monte Carlo error of primary outcomes for each optimisation metric are presented in Table 6. Effects were homogeneous across scenarios. We illustrate results specific to the scenario where $p = 8$, $EF = 0.3$, and $n = 1N$ in Figures 3 and 5. Calibration plots for the scenarios where $EF = 0.5$ are presented in Figure 4. Results for other scenarios can be visualised at [THIS WILL BE A LINK TO A SHINY APP].

Varying optimisation metrics had very limited effects on performance and computational time. Discrimination was equal across all optimisation metrics and close to 0.7, the AUC of the data when interactions are not accounted for. All optimisation metrics showed similar model calibration performance, with the exception of optimising Cohen's Kappa and classification accuracy, which showed a worse median calibration performance. On the other hand, there are no notable differences in the calibration plots between optimisation metrics (Figure 4). Runtime was comparable across all optimisation metrics. We select

TABLE 6 Average performance of optimisation metrics (aggregated over all scenarios). Rows are sorted in ascending order of RMSD(slope).

Optimisation metric	AUC	Calibration slope	RMSD(slope)	Runtime (seconds)
	Mean (SD)	Median (IQR)		Mean (SD)
Logarithmic loss	0.7 (0.02)	1.05 (0.28)	0.20	261.7 (301.4)
Calibration slope	0.7 (0.02)	0.98 (0.27)	0.21	263.0 (301.9)
Deviance	0.7 (0.02)	1.05 (0.28)	0.21	261.7 (301.4)
Brier score	0.7 (0.02)	1.03 (0.29)	0.22	261.4 (300.8)
Calibration intercept	0.7 (0.02)	1.13 (0.31)	0.23	262.9 (301.3)
AUC	0.7 (0.02)	1.04 (0.33)	0.27	262.2 (301.6)
Classification accuracy	0.7 (0.02)	0.9 (0.33)	0.30	262.0 (301.5)
Cohen's Kappa	0.7 (0.02)	0.78 (0.36)	0.42	262.1 (301.5)

TABLE 7 Average performance of hyperparameter search algorithms (aggregated over all scenarios). Rows are sorted in ascending order of runtime.

Search algorithm	AUC	Calibration slope	RMSD(slope)	Runtime (seconds)
	Mean (SD)	Median (IQR)		Mean (SD)
Model-based optimisation	0.7 (0.02)	1.07 (0.28)	0.21	70.5 (032.1)
Random search	0.7 (0.02)	1.06 (0.29)	0.22	97.4 (127.7)
Grid search	0.7 (0.02)	1.05 (0.28)	0.21	268.3 (305.8)

logarithmic loss to be optimised in Study 3, as it is among the best performing metrics and is an in-built metric in random forest optimisation packages (e.g. `tuneRanger`⁷), unlike deviance, making our research more easily applicable for model-users.

Secondary outcomes showed no substantial difference in performance (Figure 3) between optimisation metrics. We further investigated whether optimising a given metric in the training dataset led to better performance on that metric in the validation dataset. This is presented in Figure 5. For all metrics, it seems performance was not improved by optimising for that metric.

3.3 | Study 3: Hyperparameter search algorithms

Based on findings from Studies 1 and 2, we tuned `mtry` and `min.node.size`, optimising for logarithmic loss. The average and Monte Carlo error of primary outcomes for each search algorithm are presented in Table 7. Effects differed across scenarios: while model-based optimisation had comparable runtimes across all scenarios (leading to a lower overall standard deviation), runtimes for grid search and random search varied between scenarios (Figure 6). Grid search always took longer than random search. Model-based optimisation was at times longer than grid search (e.g., when $p = 8$, $EF = 0.3$, $n = 0.5N$), shorter than random search (e.g., when $p = 16$, $EF = 0.1$, $n = 1N$), or fell between grid search and random search (e.g., when $p = 8$, $EF = 0.5$, $n = 1N$). However, predictive performance was homogeneous across scenarios. We illustrate results specific to the scenario where $p = 8$, $EF = 0.3$, and $n = 1N$ in Figure 7. Calibration plots for the scenarios where $EF = 0.5$ are presented in Figure 8. Results for other scenarios can be visualised at [THIS WILL BE A LINK TO A SHINY APP].

Varying hyperparameter search algorithm did not appear to have an effect on model performance. All three algorithms showed some variation in calibration slope and intercept, but seemed to perform well overall. Random search seems to have a slightly stronger tendency to underestimate risk than grid search and model-based optimisation. The AUC showed some variation around 0.7, which is the expected AUC when not accounting for interactions in the data. Cohen's Kappa showed some variation within hyperparameter search algorithms. Brier score, logarithmic loss, and classification accuracy did not show variations.

3.4 | Study 4: Post-hoc sensitivity analysis

text here

4 | DISCUSSION

We aimed to identify optimal tuning strategies for random forests used for clinical prediction modelling in low-dimensional settings. To this end, we studied the impact of the choice of hyperparameters being tuned, the optimisation metric used, and the hyperparameter search algorithm on model performance and computational time. We ran a simulation study for each of these three choices, using 500 runs for each of twelve data simulation scenarios. Following the first study, we concluded that tuning only `mtry` and `min.node.size` led to the best performance, and strongly advise against tuning more hyperparameters, as this leads to large increases in computational time and a decrease in calibration performance. This advice is particularly strong for `splitrule`, as tuning this hyperparameter systematically led to worse calibration outcomes compared to not tuning it. Following the second study, we concluded that most optimisation metrics led to comparable performances on all outcomes. We advise against using classification accuracy and Cohen's Kappa as optimisation metrics as these had slightly worse calibration performances. Due to existing software implementations, we opted to optimise logarithmic loss in our final study. We concluded that grid search, random search, and model-based optimisation all led to models with comparable predictive performance. Computational time heavily depended on sample size and the number of iterations for grid search and random search. Model-based optimisation had similar computational times across scenarios. While we do not advise against the use of any of these three hyperparameter search algorithms, we suggest the use of model-based optimisation via `tuneRanger`⁷ as this, by default, tunes optimising for logarithmic loss rather than accuracy, and does not require the definition of a grid. [SOME COMMENTS ABOUT THE SENSITIVITY ANALYSIS]

We extend findings by Probst and colleagues¹² by showing that considering larger combinations of hyperparameters does not improve performance. Additionally, we provide previously lacking empirical findings advising against tuning `splitrule`. We also provide previously lacking comparisons between optimisation metrics and advise against using optimisation metrics based on predicted class rather than predicted risk. Finally, we extend findings by Yang and Shami¹³ which showed grid search to needlessly increase computational time compared to other search algorithms as it did not improve classification accuracy. Our results complement this by showing grid search does not improve more clinically relevant, performance metrics, such as calibration and discrimination. There are some caveats to this finding: the computational time of grid search is heavily dependent on the size of the grid (and, similarly, the computational time of random search is dependent on the number of iterations), making it occasionally more computationally efficient than model-based optimisation. However, grid search requires the researcher to not only select which hyperparameters are tuned and which metric is optimised, but also the range and step size of numerical candidate hyperparameters, making it, more often than not, an inefficient algorithm involving a larger number of subjective choices than model-based optimisation. While a similar point could be made regarding the number of iterations of model-based optimisation, the application of this algorithm as computed by `tuneRanger`⁷ was developed such that default values for the number of iterations lead to a relatively short computational time with good final predictive performance. Our findings thus reinforce findings showing model-based optimisation to be among the better search algorithms^{7,13}, and provide additional evidence that overall simpler tuning procedures lead to better outcomes.

4.1 | Limitations and future outlook

We provide an overview of possible variables in tuning procedures; however, we only provide limited insight into interactions these may have. While it is currently not computationally feasible to conduct a full factorial study, it is important to keep in mind that as our studies incorporated previous results, this may have introduced some bias in our conclusions. This may be exacerbated by the fact that variability was reduced by simulating all data under logistic regression. Tree-based methods are popular in part due to their flexibility (CITATION), such that they are appropriate for a wide variety of data. Data-generating mechanisms may have a notable impact on relevant tuning procedures; however, our results fit well with research conducted on empirical datasets considering the choice of hyperparameters to tune¹² or the hyperparameter search algorithm¹³. We thus consider it unlikely that our findings were heavily impacted by the choice of data-generating mechanism.

We did not study the effect of hyperparameter candidate values to build a grid, or the number of iterations in random search. These two search algorithms' performance are heavily dependent on user input. While our findings show them to have comparable performance to model-based optimisation, future research may show this is only the case in optimal settings. Varying the input for the grid may lead to worse predictive performances, such that future research may show model-based optimisation to not only be on par with grid search, but better with regards to model performance.

4.2 | Conclusion

Random forests are widely used in clinical research^{3,4}, yet can suffer from miscalibration^{9,10}. We examined the impact of various tuning procedures on random forests' predictive performance for dichotomous risk predictions and on computational time. We show simpler hyperparameter tuning procedures lead to better clinically relevant outcomes without needlessly inflating computational time.

References

1. Wessler BS, Paulus J, Lundquist CM, et al. Tufts PACE clinical predictive model registry: update 1990 through 2015. *Diagnostic and prognostic research* 2017; 1(1): 1–8.
2. Mitchell TM, Mitchell TM. *Machine learning*. 1 . 1997.
3. Uddin S, Khan A, Hossain ME, Moni MA. Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making* 2019; 19(1): 281.
4. Andaur Navarro CL, Damen JAA, Smeden vM, et al. Systematic Review Identifies the Design and Methodological Conduct of Studies on Machine Learning-Based Prediction Models. 2022; 154: 8–22. doi: 10.1016/j.jclinepi.2022.11.015
5. Mantovani RG, Horváth T, Cerri R, Junior SB, Vanschoren J, Carvalho dACPdLF. An empirical study on hyperparameter tuning of decision trees. 2019. doi: 10.48550/arXiv.1812.02207
6. Bernard S, Heutte L, Adam S. *Influence of Hyperparameters on Random Forest Accuracy*. Lecture Notes in Computer Science . 2009
7. Probst P, Wright MN, Boulesteix AL. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery* 2019; 9(3): e1301. doi: 10.1002/widm.1301
8. Van Calster B, McLernon DJ, Smeden vM, et al. Calibration: the Achilles heel of predictive analytics. *BMC Medicine* 2019; 17(1): 230.
9. Benedetto U, Sinha S, Lyon M, et al. Can Machine Learning Improve Mortality Prediction Following Cardiac Surgery?. 2020; 58(6): 1130–1136. doi: 10.1093/ejcts/ezaa229
10. Djulbegovic B, Berano Teh J, Wong L, Hozo I, Armenian SH. Diagnostic Predictive Model for Diagnosis of Heart Failure after Hematopoietic Cell Transplantation (HCT): Comparison of Traditional Statistical with Machine Learning Modeling. 2019; 134: 5799. doi: 10.1182/blood-2019-130764
11. Strubell E, Ganesh A, McCallum A. Energy and Policy Considerations for Deep Learning in NLP. 2019. doi: 10.48550/arXiv.1906.02243
12. Probst P, Boulesteix AL, Bischl B. Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research* 2019; 20(1): 1934–1965.
13. Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 2020; 415: 295–316. doi: 10.1016/j.neucom.2020.07.061
14. Ellenbach N, Boulesteix AL, Bischl B, Unger K, Hornung R. Improved Outcome Prediction Across Data Sources Through Robust Parameter Tuning. *Journal of Classification* 2021; 38(2): 212–231. doi: 10.1007/s00357-020-09368-z
15. Morris TP, White IR, Crowther MJ. Using simulation studies to evaluate statistical methods. *Statistics in Medicine* 2019; 38(11): 2074–2102. doi: 10.1002/sim.8086
16. Bischl B, Casalicchio G, Feurer M, et al. OpenML Benchmarking Suites. 2021(arXiv:1708.03731). doi: 10.48550/arXiv.1708.03731
17. Riley RD, Ensor J, Snell KIE, et al. Calculating the sample size required for developing a clinical prediction model. *BMJ* 2020; 368: 441. doi: 10.1136/bmj.m441
18. Wright MN, Ziegler A. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software* 2017; 77(1): 1–17. doi: 10.18637/jss.v077.i01
19. Kuhn M. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software, Articles* 2008; 28(5): 1–26. doi: 10.18637/jss.v028.i05

-
- 20. Probst P, Boulesteix AL. To Tune or Not to Tune the Number of Trees in Random Forest. *Journal of Machine Learning Research*: 18.
 - 21. Luijken K, Groenwold RHH, Van Calster B, Steyerberg EW, Smeden vM. Impact of predictor measurement heterogeneity across settings on the performance of prediction models: A measurement error perspective. *Statistics in Medicine* 2019; 38(18): 3444–3459. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.8183> doi: 10.1002/sim.8183
 - 22. Van Calster B, Smeden vM, De Cock B, Steyerberg EW. Regression shrinkage methods for clinical prediction models do not guarantee improved performance: Simulation study. *Statistical Methods in Medical Research* 2020; 29(11): 3166–3178. doi: 10.1177/0962280220921415
 - 23. R Core Team . R: A Language and Environment for Statistical Computing. 2023. version 4.2.3.



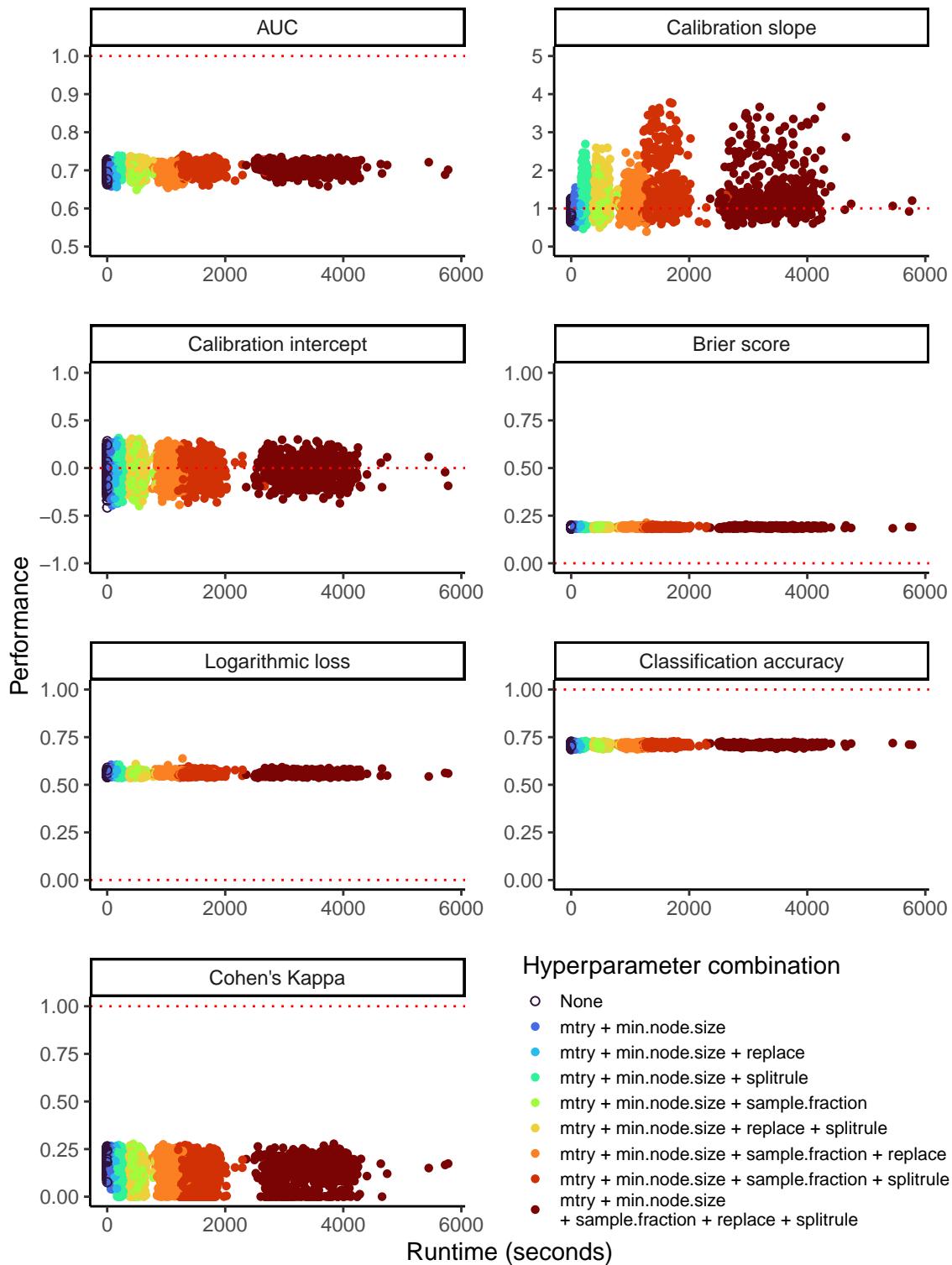


FIGURE 1 Performance of each hyperparameter tuning combination in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$. Red dotted lines show ideal performance.

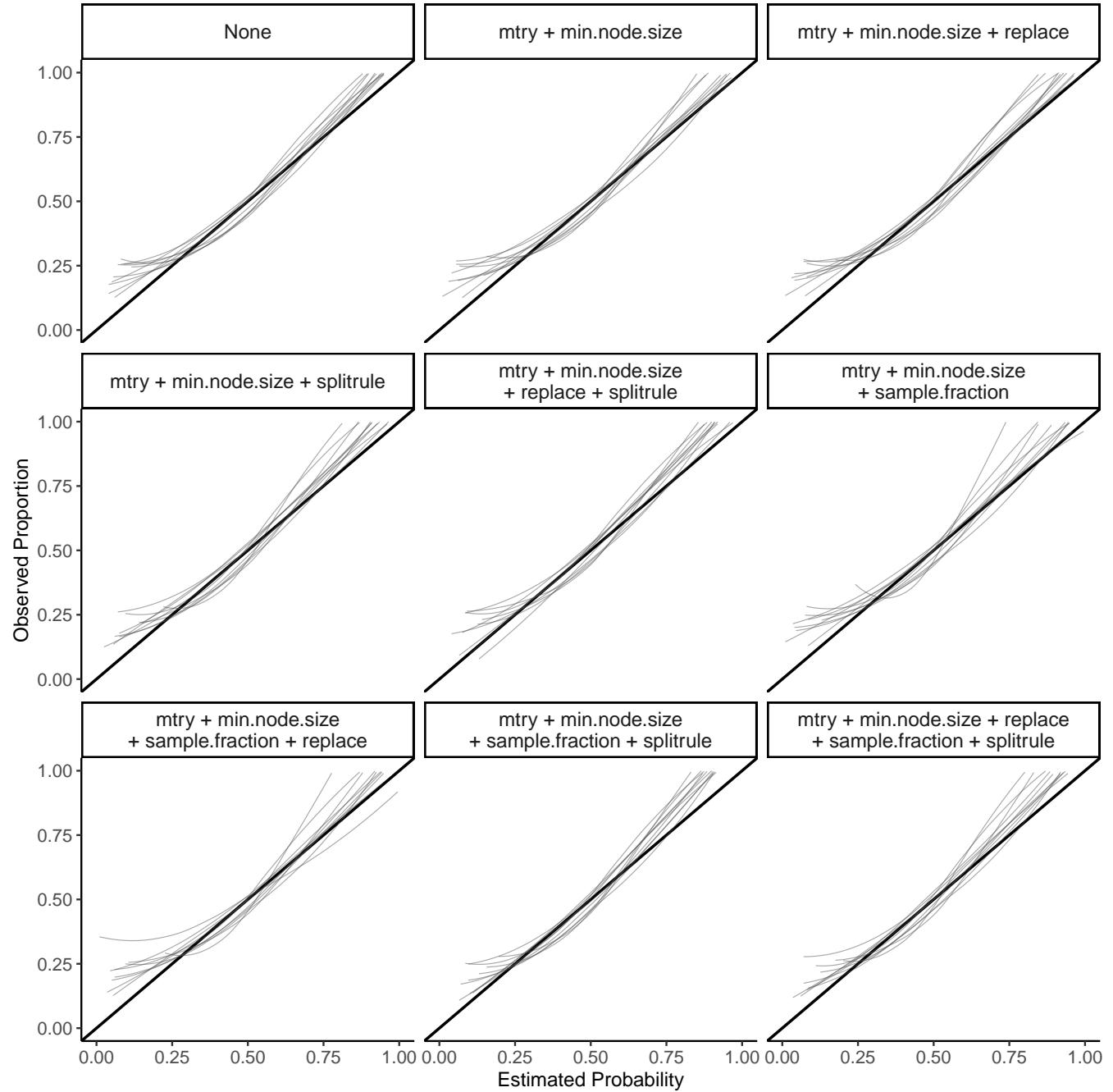


FIGURE 2 Calibration plots comparing hyperparameter combinations for every tenth dataset in scenarios where $EF = 0.5$

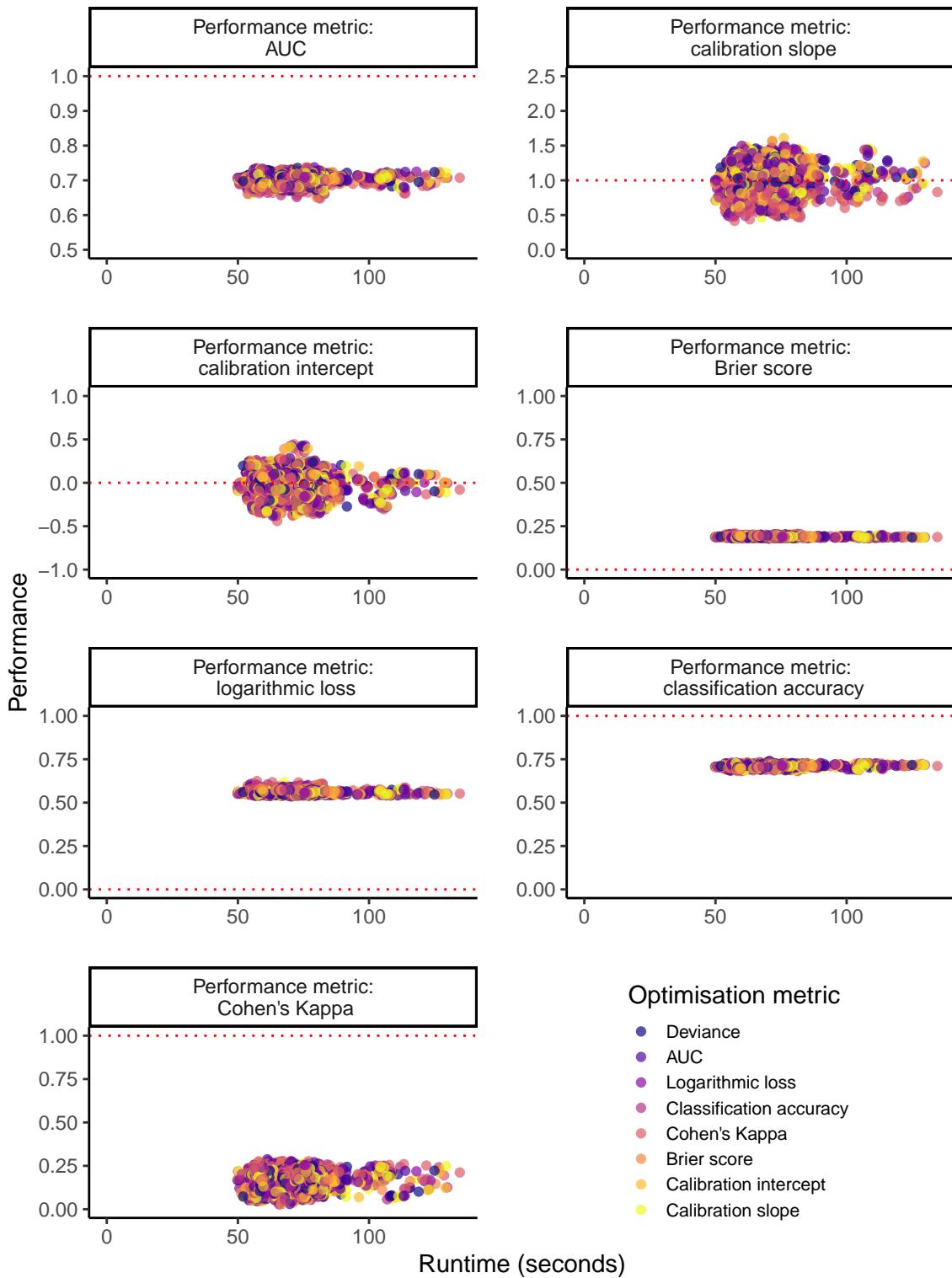


FIGURE 3 Performance of each optimisation metric in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$. Red dotted lines show ideal performance.

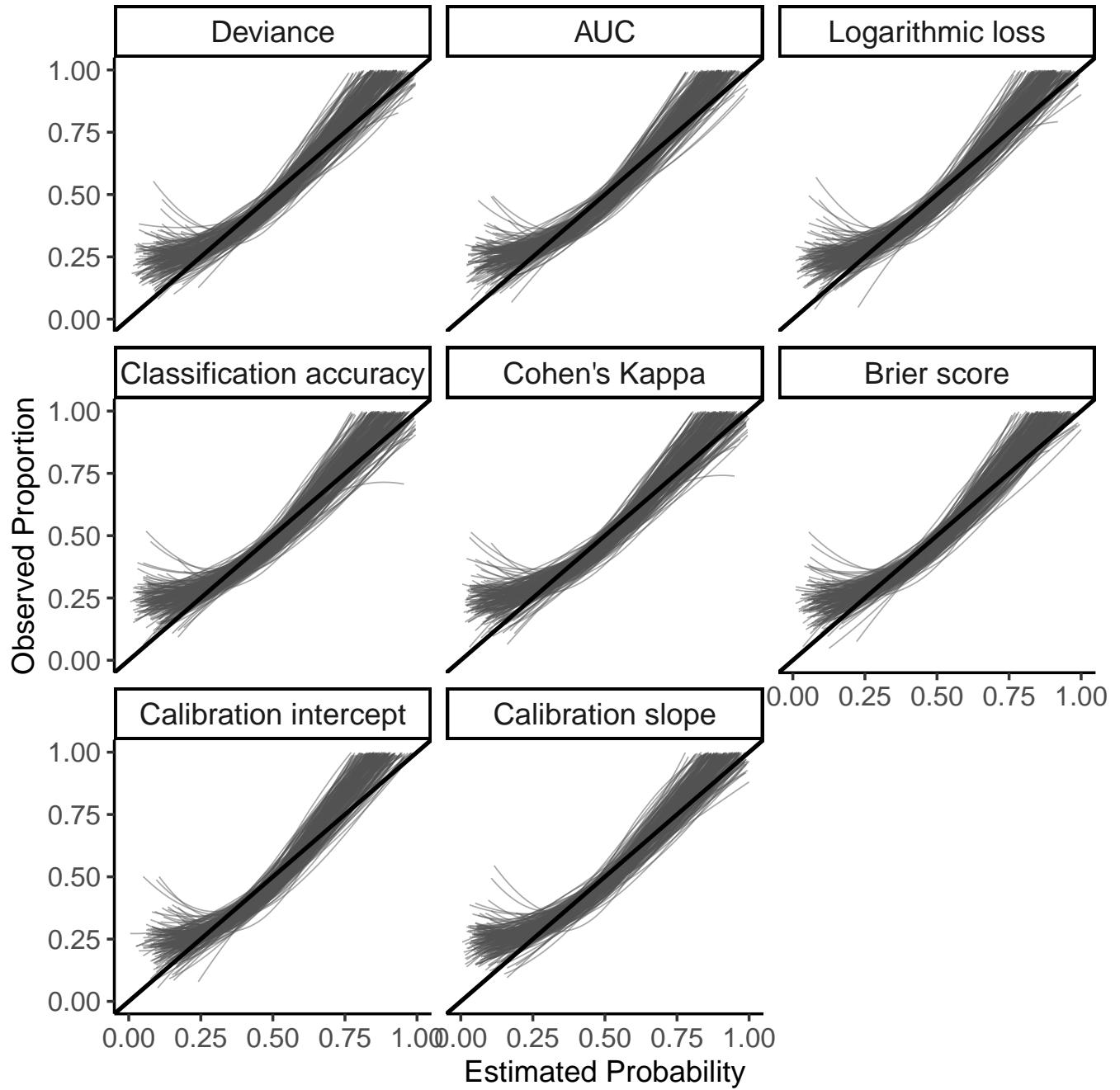


FIGURE 4 Calibration plots comparing optimisation metrics for every tenth dataset in scenarios where $EF = 0.5$.

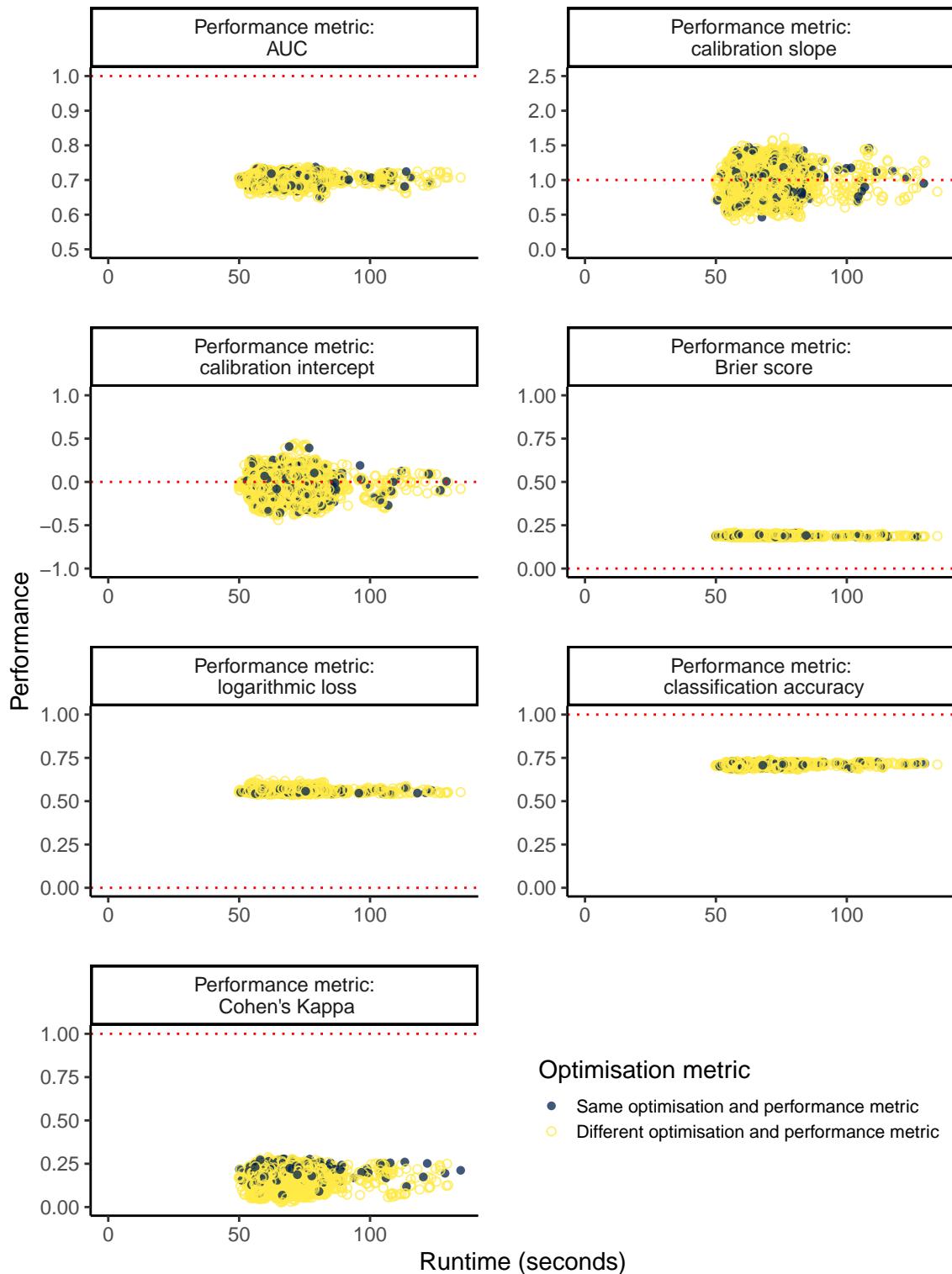


FIGURE 5 Comparison of the performance of optimisation metrics and corresponding performance metrics in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$. Red dotted lines show ideal performance.

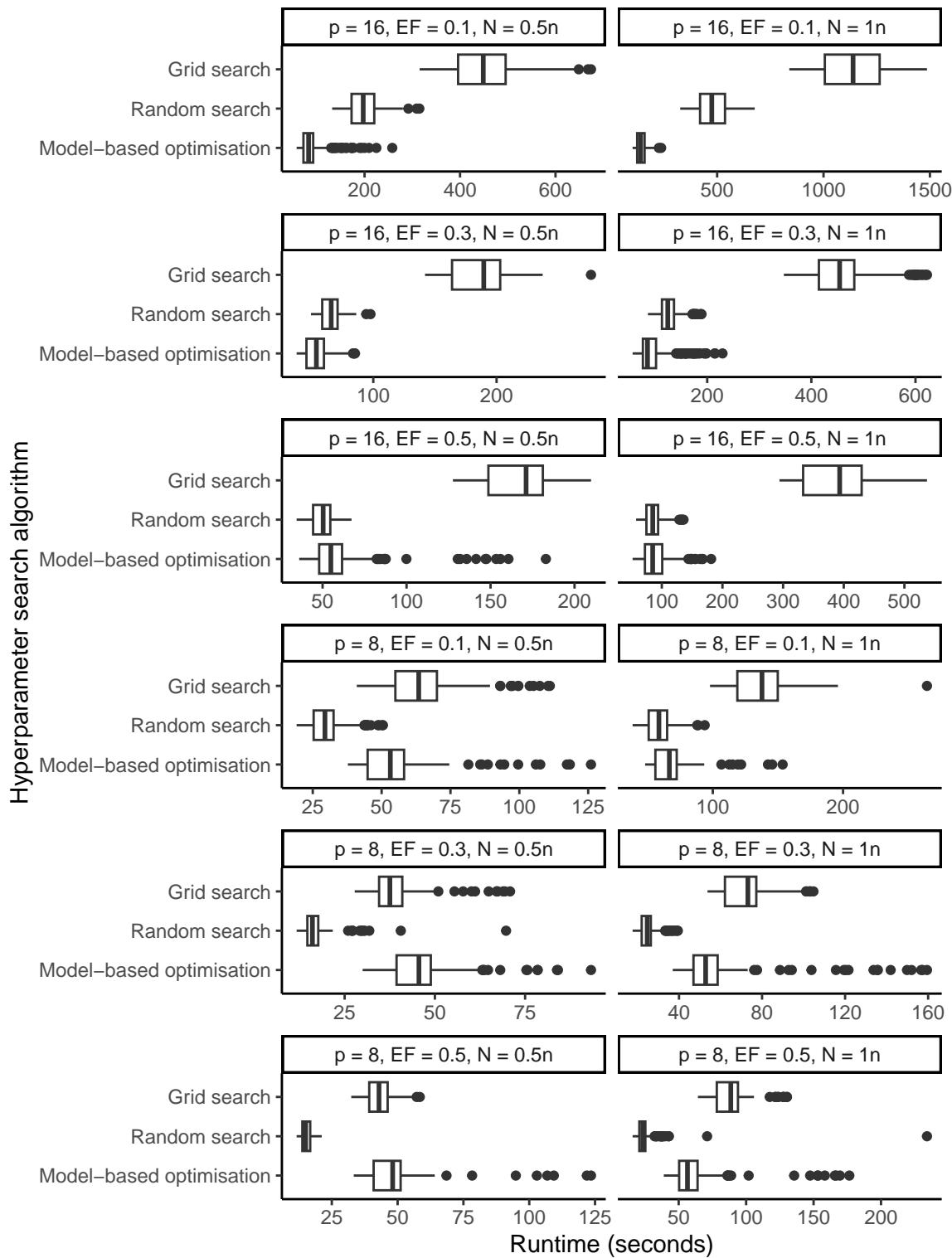


FIGURE 6 Comparison of runtimes for the hyperparameter search algorithms between scenarios

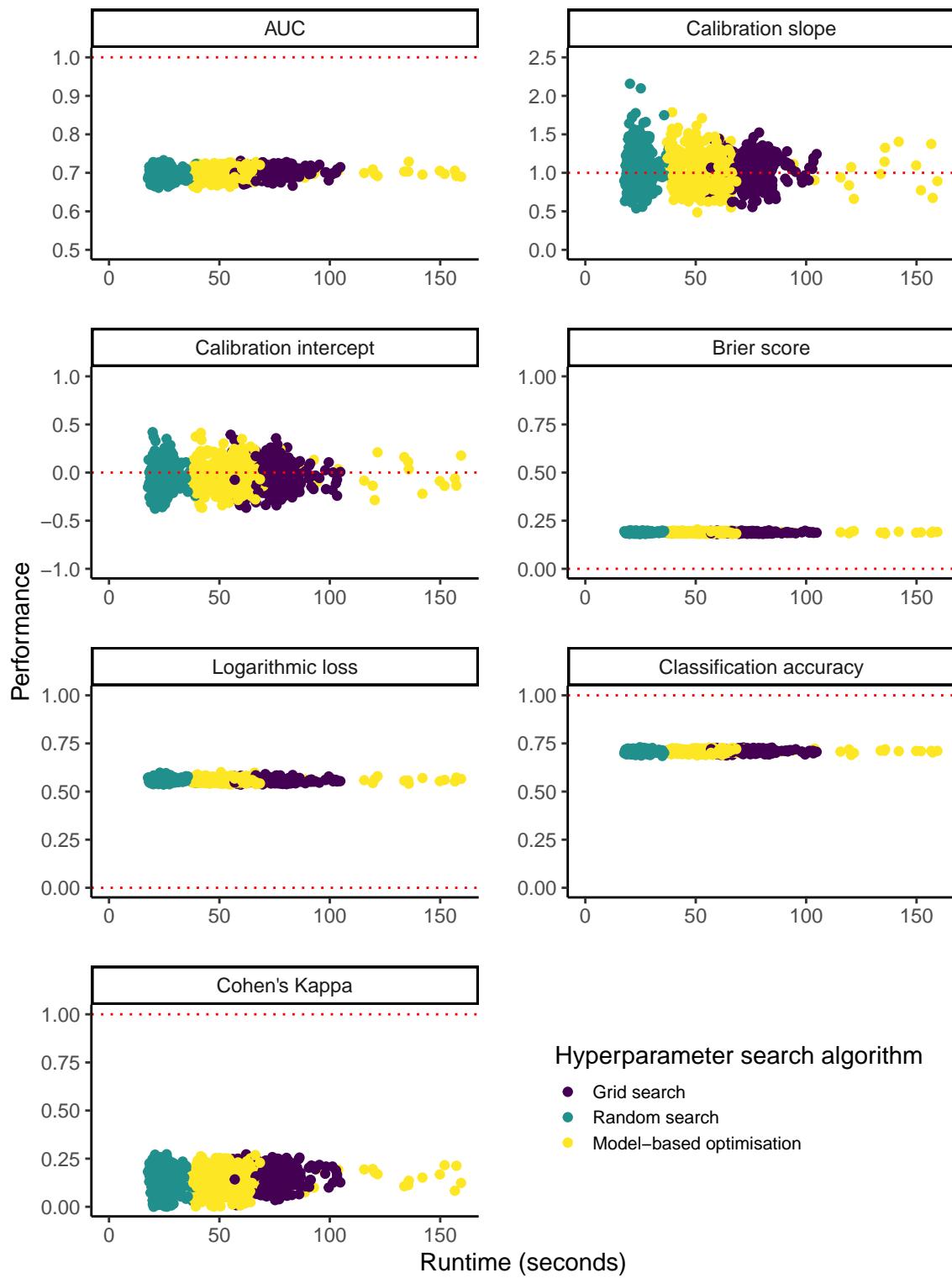


FIGURE 7 Performance of each hyperparameter search algorithm in the example scenario where $p = 8$, $EF = 0.3$, $n = 1N$. Red dotted lines show ideal performance.

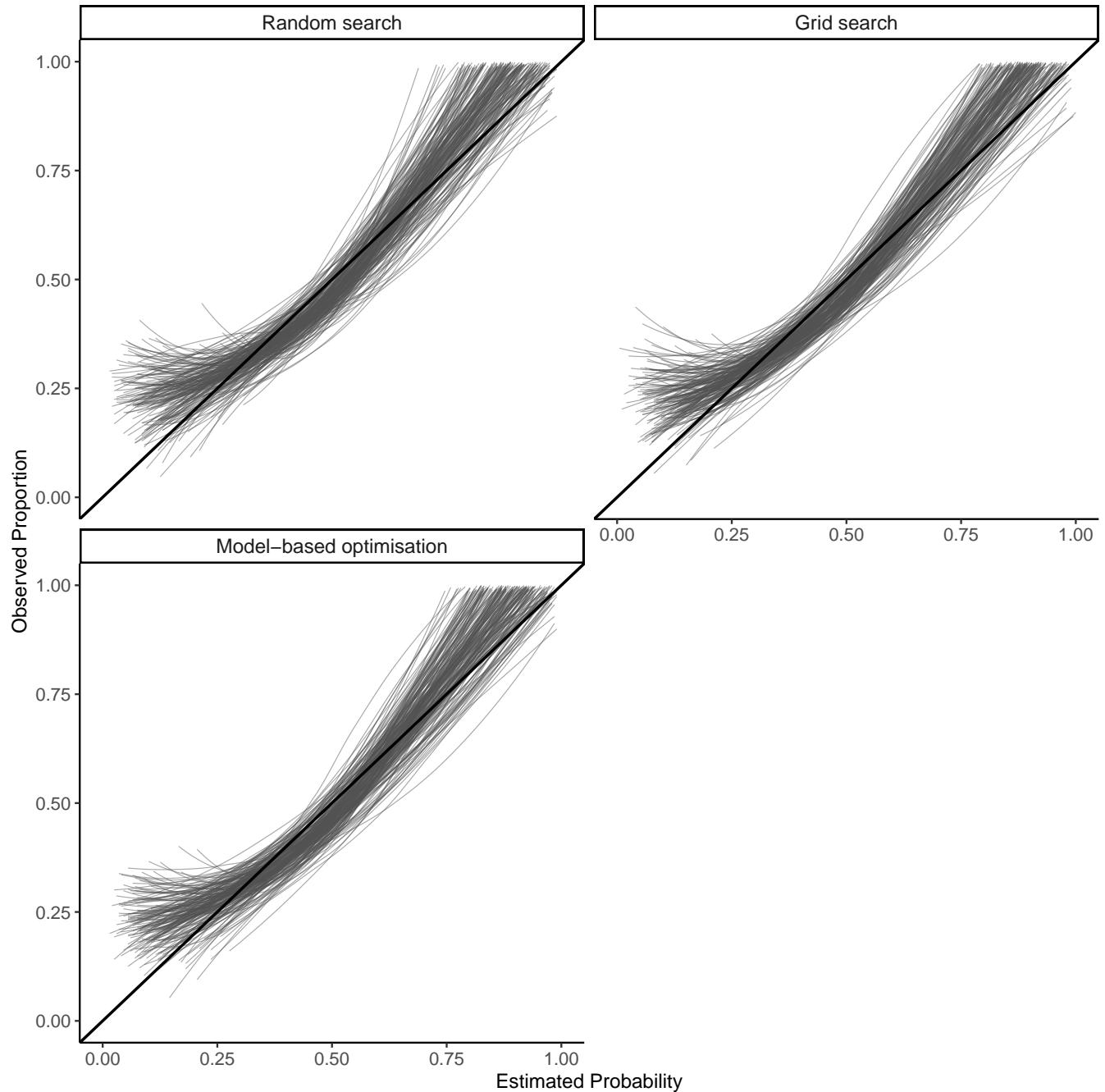


FIGURE 8 Calibration plots comparing hyperparameter search algorithms for every tenth dataset in scenarios where $EF = 0.5..$