Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Performance of clinical prediction models versus computational time for random forest hyperparameter tuning procedures: A simulation study

J.A. Neve*[1,2]

[1]Department of Methodology and Statistics, Utrecht University, Utrecht, Netherlands
[2]Julius Centre, University Medical Centre Utrecht, Utrecht, Netherlands

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

Hyperparameter tuning is a well-known way to improve predictive performance. Tuning procedures vary in their computational demands, with no clear relation to model predictive performance. In this study, we identified optimal hyperparameter tuning strategies for balancing predictive performance and computational time of random forests for clinical prediction models with a binary outcome in low-dimensional settings. We conducted three sequential simulation studies to systematically compare the calibration, discrimination, and computational time of sets of (i) hyperparameters to tune, (ii) optimisation criteria, and (iii) hyperparameter search algorithms. Results showed varying which hyperparameters are tuned led to important differences in predictive performance and computational time. Computational time varied from a few seconds to multiple hours. Larger combinations of tuned hyperparameters were associated with increased computational time and decreased calibration performance. Varying optimisation criteria led to some differences in predictive performance, but not computational time. Predicted class-based criteria performed worse than predicted risk-based criteria. Hyperparameter search algorithms varied in computational time, but not predictive performance. The shortest search algorithm differed between data-generating scenarios. Search algorithms also showed different levels of sensitivity to sample size, with model-based optimisation having the smallest increase in runtime when sample size doubled. We conclude the optimal tuning procedure was to tune the number of randomly sampled predictors at a split and the minimum node size using model-based optimisation, minimising logarithmic loss, and highlight the need to carefully consider tuning procedures in practice.

KEYWORDS:
Hyperparameter tuning; random forests; machine learning; biostatistics; predictive modelling

1 | INTRODUCTION

Tree-based methods[1] are increasingly used in medicine for diagnostic and prognostic purposes[2]. Random forests are particularly prevalent[3,4]. Random forests' results are known to be influenced by hyperparameters (e.g., the number of predictors sampled to make a split)[5] and the way they are tuned (e.g., using a cross-validation based grid search)[6]. 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[7,8]. Classification accuracy is however not sufficient to evaluate a model's clinical utility: identifying a patient as "positive" or "negative" (e.g., whether or not the patient has a given disease) may not carry sufficient information, as the patient's predicted risk is a key element of medical decision-making[9,10] (e.g., the choice of treatment to undergo). Clinical prediction models are often evaluated using discrimination (i.e., positive patients have higher risk predictions than negative patients) and calibration (i.e., risk predictions reflect patients' true risk)[11]. Models with low discrimination have limited clinical utility as they are likely to fail to differentiate between "positive" and "negative" patients, while miscalibrated models can to lead over- or underestimating patients' risks. These issues can lead to to over- or undertreatment. A few clinical studies have observed random forests to heavily suffer from miscalibration[12,13].

A well-known way to improve random forest performance is to perform hyperparameter tuning. Tuning identifies hyperparameter values for which a performance metric (e.g., logarithmic loss) is optimal for a given prediction model. This procedure requires choosing which metric is optimised and how, which hyperparameters are tuned, and how candidate hyperparameter values are generated. It is possible tuning procedures improve certain aspects of performance at the cost of miscalibration. Besides the impact on predictive performance, the choice of tuning procedure can have a substantial impact on the computational time needed to estimate the model. For instance, the more hyperparameters are tuned or the more candidate values there are, the larger computational time is. This also depends on the choice of hyperparameter search algorithms, which 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)[8]. 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, the carbon emission of a single run of a model can be as much as five times the average emissions of a car over its lifetime[14].

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 tree[6]. Yang and Shami[15] 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 criteria's 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 prediction modeling typically uses low-dimensional datasets (datasets with more observations than features), for which guidance regarding hyperparameter tuning is lacking[16]. In this paper we therefore aimed to identify the optimal hyperparameter tuning strategies for balancing predictive performance and computational time of random forests in low-dimensional settings with a binary outcome. There are three main simulation studies in this project to identify (i) which hyperparameters to tune, (ii) which metric to optimise and how, and (iii) which search algorithm to use for optimal model performance.

The following sections will describe the simulation studies and the conclusions we drew from them in more detail. Section 2 covers the simulation methods for each of the three studies. Results for each study are presented in Section 3. Section 4 summarises and contextualises our findings.

2 | METHODS

We conducted three simulations studies sequentially to identify optimal tuning strategies in controlled conditions. In this section we detail the simulation methods for each study, following the ADEMP approach[17].

2.1 | Aim

2.1.1 | Study 1: Selecting hyperparameters

Using datasets from the OpenML platform[18], Probst and colleagues[6] found the number of predictors considered at a split and the sample fraction used to fit a single tree 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 model calibration in addition to model discrimination and by

considering a larger set 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 predictive performance in context of required computational times.

2.1.2 | Study 2: Optimisation criterion

The choice of metric to optimise and how it is optimised (hereafter called "optimisation criterion") may have a large effect on the results, improving certain aspects of predictive performance (e.g., discrimination) at the cost of others (e.g., calibration). To date, this has not been studied, and applied studies rarely report the optimisation criterion that was used in their tuning procedure. In this study, we aimed to examine the extent to which the choice of optimisation criterion affects different aspects of predictive performance and identify an optimisation criterion leading to the best overall 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

Yang and Shami[15] showed important differences in computational time and classification accuracy between the way hyperparameter candidate values are generated (hereafter called "hyperparameter search algorithms"). This study aimed to compare hyperparameter search algorithms with respect to random forests' predictive performance and computational time. We tuned the combination of hyperparameters considered optimal in Study 1 using the optimisation criterion selected in Study 2.

2.2 | Data-generating mechanism

All studies used the same data-generating mechanism. Datasets were generated separately for each study. A full factorial simulation design was used to evaluate the influence of data generating characteristics on the variations in model predictive performance and computational time. The outcome was a binary variable with classes "positive" and "negative". The varying simulation factors were the number of predictor variables p (range: 8, 16), the proportion of observations that had "positive" as their outcome (hereafter called "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 colleagues[19]. A total of 12 (2*3*2) scenarios were considered. For each scenario, 500 training datasets and 500 validation datasets were generated per study, yielding a total of 6,000 observations per study. Scenario sample sizes are detailed in Table 1.

Training and validation datasets were simulated under a logistic model with two-way interactions. For each observation i ($i = 1, \ldots, N$), 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 $p/4$ two-way interactions, with the h^{th} ($h = 1, ..., p/4$) 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):

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

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". The computation procedure of the true effects is presented in Appendix A.

2.3 | Simulation methods

2.3.1 | Study 1: Selecting hyperparameters

We varied which hyperparameters were tuned when fitting a random forest model using the R package `ranger`[20] via the R package `caret`[21]. We optimised logarithmic loss using grid search in a 5-fold cross-validation. Individual hyperparameter definitions are presented in Table 2. We refer to hyperparameters using the notation in `ranger`.

Random forests can have up to ten hyperparameters[20], some of which are only used if another hyperparameter has a certain value (e.g., there 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

focused on hyperparameters included in a previous study by Probst and colleagues[6], which focused on hyperparameters' effects on predictive performance in order to expand the findings by considering hyperparameters' combined effects. This led to the inclusion of `mtry`, `min.node.size`, `replace`, and `sample.fraction`. Although 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, number[22]. Additionally, we included `splitrule` as it can be tuned by commonly used software such as `caret`[21] (which does not tune certain other hyperparameters such as `replace`). Hyperparameter default values and tuning ranges used in this study are presented in Table 2.

Considering all possible combinations of these five hyperparameters still leads to an unfeasible number of 31 combinations. We used Probst and colleagues' results[6] 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 similar effects. The commonly used tuning package `caret`[21] 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. A model using the default hyperparameters was fit to establish the baseline. This yielded 9 hyperparameter combinations, including the baseline model. Hyperparameters not included in a given combination were set to their default value. Each considered combination was used to fit a random forest on each training dataset.

2.3.2 | Study 2: Optimisation criterion

We varied the optimisation criterion used when fitting a random forest using the R package `ranger`[20] via the R package `caret`[21]. Each dataset was tuned once using each candidate optimisation criterion. We used grid search in a 5-fold cross-validation to tune the hyperparameters considered optimal in Study 1. Tuning ranges were identical to those of Study 1 (Table 2). The considered candidate optimisation criteria were minimising the logarithmic loss (Formula 2; measuring the mean deviations between predicted risk and the true outcome), maximising the AUC (a measure of how well the classes are separated), minimising the Brier score (Formula 3; this can be seen as a composite measure of calibration and discrimination[23]), maximising the classification accuracy (the proportion of correctly classified observations) and maximising Cohen's Kappa (the proportion of correctly classified observations, adjusted for chance).

$$\text{Logarithmic loss} = -\frac{1}{n}\sum_{i=1}^{n}(y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)), \tag{2}$$

$$\text{Brier score} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3}$$

Additionally, we developed two candidate optimisation criteria focusing on calibration, which is not typically optimised. Calibration can be measured in various ways[11]. We focused on calibration intercept and calibration slope. Calibration intercept is a measure of the difference between the average predicted risk and the true event rate. It is calculated by estimating the intercept of a logistic regression model predicting the observed outcomes from the recalibrated predicted risks (Formula 4). The optimal value for calibration intercept is 0, with negative values suggesting overestimation and positive values suggesting underestimation. Calibration slope measures the spread of predicted risks to assess whether they may be systematically too extreme or too moderate. This is calculated by estimating the logistic regression model presented in Formula 5 and defining the calibration slope to be the slope coefficient in the model. The optimal value for calibation slope is 1, with values under 1 indicating risks are too extreme and values over 1 indicating risks are too moderate. For our calibration intercept-based optimisation criterion, we chose to minimise the square of the calibration intercept in order to penalise both under- and overestimation. For our calibration slope-based optimisation criterion, we chose to minimise the square of the natural logarithm of calibration slope in order to penalise too-extreme risks more than too-moderate risks.

$$\log\frac{\mathbf{y}}{1-\mathbf{y}} = \beta_0 + \log\frac{\hat{\mathbf{y}}}{1-\hat{\mathbf{y}}}, \tag{4}$$

where $\mathbf{y}$ represents the vector of observed binary outcomes, β_0 represents the intercept term in the model, and $\hat{\mathbf{y}}$ represents the vector of predicted risks. This is an intercept-only model.

$$\log\frac{\mathbf{y}}{1-\mathbf{y}} = \beta_0 + \beta_1\log\frac{\hat{\mathbf{y}}}{1-\hat{\mathbf{y}}}, \tag{5}$$

where $\mathbf{y}$ represents the vector of observed binary outcomes, β_0 represents the intercept term in the model, β_1 represents the slope coefficient, and $\hat{\mathbf{y}}$ represents the vector of predicted risks.

2.3.3 | Study 3: Hyperparameter search algorithm

We varied the hyperparameter search algorithm used when fitting a random forest. We tuned the optimal hyperparameters selected from Study 1 and used the optimisation criterion that performed optimally in Study 2. Grid search, random search (both using the R package `caret`[21]), and model-based optimisation (using the R package `tuneRanger`[8]) were used.

Grid search takes a fixed set of values for each hyperparameter and fits a random forest using each possible combination of these hyperparameter values. 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 with the best value according to the optimisation criterion is selected to be the optimal hyperparameter values. These two algorithms were performed using 5-fold cross-validation. Grid search used a custom 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 custom grid with `mtry` going from 1 through to p and `min.node.size` going from 1 through to $EF * n$, then randomly selecting a number of rows equal to $5p$ (such that the number of iterations was half the number of iterations in grid search).

Model-based optimisation first computes a given number (here set to the software default of 30) of random combinations of hyperparameter values and uses each of these combinations to fit a random forest and assess its performance using a given performance metric. Then, it fits a regression model with the hyperparameter values as predictors and the performance as the outcome to identify a combination of hyperparameters that is expected to perform better. This combination is tested, and the regression model is fit again including the new point. This is repeated a given number of times (here set to the software default of 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 colleagues[8] for further details.

2.4 | Performance measures

The same primary outcomes were used in all studies. Primary outcomes were defined as outcomes which were used to advise on the best tuning procedures. Discrimination was measured by the AUC and assessed whether observations with the event have higher predicted risk observations without the event. Calibration was measured by the calibration slope and assessed the extent to which predicted risks reflect true risks. The root mean square of the log of the calibration slope over all the runs of a scenario (RMSD(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 colleagues[24]. Computational time was the number of seconds for which a given tuning procedure ran.

Performance metrics used as part of the optimisation criteria in 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.

Each model built using the training data was used to predict risks on an independently generated validation dataset ($n = 100,000$ for Study 1, reduced to $n = 10,000$ for Studies 2 and 3 due to storage contraints) in order to evaluate model performance. For each tuning procedure, we computed the average and Monte Carlo error of each of the primary performance measures. We visualised whether certain hyperparameter combinations, optimisation criteria, or hyperparameter search algorithms had a notably larger runtime compared to others, without showing a substantial increase in performance as measured by both the primary and secondary outcomes. The optimal hyperparameter combination, optimisation criterion, and hyperparameter search algorithm were assessed considering all primary outcomes.

2.5 | Software

Data was simulated and tuning procedures were performed on the high-performance computer of the University Medical Center Utrecht, which has two types of processors (Xeon Gold and Xeon E5). This was done in R version 4.2.2, using packages `MASS` version 7.3-58.1[25], `dplyr` varsion 1.0.10[26], `ranger` version 0.14.1[20], `pROC` version 1.18.0[27], `psych` version 2.2.9[28], `caret` version 6.0-93[21], and in the case of Study 3, `tuneRanger` version 0.5[8]. Summary statistics and figures were generated on a personal computer using R version 4.2.3[29] using packages `tidyverse` version 2.0.0[30], `viridis` version 0.6.2[31], `xtable` version 1.8-4[32], and `rticles` version 0.24[33].

2.6 | Error handling and reproducibility

All code used to generate and analyse data is available at https://github.com/judithneve/HyperparameterTuning.

Where performance metrics used the natural logarithm of risk predictions, values of 0 and 1 were recoded to 1e-16 and 1-1e-16, respectively, in order to avoid missing values in computation. Simulation runs could fail due to reaching the maximum allocated time. Details of allocated times and failures are found in Table S1 in the supplementary materials.

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 in parallel, so it can never be reproduced identically. Additionally, the two types of processors of the high-performance computer responded differently to seeds. It could not be controlled which runs were assigned to which operating systems. 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.

2.7 | Ethics

This study was granted ethical approval by Utrecht University's ethics committee. It is filed under number 22-1808.

3 | RESULTS

Results were often consistent across different values of number of predictor variables, event fraction, or sample size. Where that is the case, we state we visualise results for only one simulation scenario. Visualisations for other scenarios can be found at https://judithneve.shinyapps.io/HyperparameterTuning/.

3.1 | Study 1: Selecting hyperparameters

The results of primary outcomes for each hyperparameter combination are summarised in Table 3. Simulation scenario-specific summary statistics are presented in Tables S2-S4 in the supplementary materials. Effects did not meaningfully vary between different values for number of predictors, event fraction, or sample size. We illustrate results specific to the scenario where $p = 16$, $EF = 0.5$, and $n = 1N$ in Figure 1. Calibration plots for all scenarios where $EF = 0.5$ are presented in Figure 2. Calibration plots for other scenarios can be found in the supplementary materials (Figures S1 and S2).

Model calibration varied considerably within (Figure 1) and between (Table 3, Figure 1, Figure 2) combinations. On average, tuning larger combinations of hyperparameters led to worse calibration performance than tuning smaller combinations of hyperparameters, with some of the larger hyperparameter combinations performing worse than using all default hyperparameters. The best calibration performance was obtained by tuning the combination of `mtry`, `min.node.size`, and `replace`, which had a median calibration slope of 1.03 across all scenarios. This performance was among the most stable with an IQR of 0.27. The worst calibration performances were observed when `splitrule` was included in a hyperparameter combination. Including `splitrule` in any combination led to a median calibration slope above 1.13 and an RMSD(slope) above 0.26, while combinations without `splitrule` all had median calibration slopes below 1.09 and RMSD(slope) below 0.24. Including `replace` in any combination was associated to a slightly better calibration performance than the calibration performance of the same combination without `replace`. Discrimination was slightly better in combinations including `splitrule` than in combinations not including it, but this effect was very small, comparing a mean AUC of 0.71 (SD = 0.02) to a mean AUC of 0.70 (SD = 0.02). There was no noticeable difference between hyperparameter combinations in other performance metrics (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.

Computational time increased drastically the more hyperparameters were tuned, going from an average of 4.63 minutes (SD = 5.21) when tuning only `mtry` and `min.node.size` to 3.11 hours (SD = 2.99) when tuning all five included hyperparameters. In the most computationally intensive scenario (Table S2), this was a difference of 19.23 minutes (SD = 2.58) to 10.70 hours (SD = 1.84).

Balancing predictive performance and computational time, we select `mtry` and `min.node.size` as the optimal hyperparameter combination to tune, as it had an average predictive performance very similar to that of `mtry`, `min.node.size`, and `replace`, but less than half its computational time (19.23 minutes compared to 44.56 minutes in the most computationally expensive scenario).

3.2 | Study 2: Optimisation criterion

Based on Study 1, we tuned `mtry` and `min.node.size` as they were found to be optimal in terms of computational time with similar or better predictive performance compared to other sets of hyperparameters. The results of primary outcomes for each optimisation criterion are summarised in Table 4. Simulation scenario-specific summary statistics are presented in Tables S5-S7 in the supplementary materials. The effect of optimisation criterion did not meaningfully vary between different values for number of predictors, event fraction, or sample size. We illustrate results specific to the scenario where $p = 16$, $EF = 0.5$, and $n = 1N$ in Figure 3. Calibration plots for all scenarios where $EF = 0.5$ are presented in Figure 4. Calibration plots for other scenarios can be found in the supplementary materials (Figures S3 and S4).

Varying the optimisation criterion had very limited effects on predictive performance. Model calibration was similar across all optimisation criteria, with the exception of optimising for Cohen's Kappa or classification accuracy, which showed a worse median calibration performance. These differences were not noticeable in the calibration plots (Figure 4). Average discriminative performance was equal across all optimisation criteria with an AUC of 0.70. There was no noticeable difference between optimisation criteria in other performance metrics (Figure 3). We further investigated whether using a given optimisation criterion in the training dataset led to better performance on the corresponding performance metric in the validation dataset (Figure 5). Using an optimisation criterion focused on a given metric did not lead to a visibly better performance on that metric in the validation dataset compared to using optimisation criteria focused on other metrics.

Computational time was extremely similar across all optimisation criteria. Most of the variation in computational time was accounted for by the simulation scenario, with a mean runtime ranging from 38.63 seconds (SD = 6.74) when $p = 8$, $EF = 0.3$, $n = 0.5N$ to 18.72 minutes (SD = 2.71) when $p = 16$, $EF = 0.1$, $n = 1N$.

There is no "optimal" optimisation criterion balancing computational time and predictive performance as most criteria perform equally. We select logarithmic loss to be minimised in Study 3, as it is among the best performing metrics and is an in-built criterion in random forest optimisation packages (e.g. `tuneRanger`[8]), making our research more easily applicable for model-users.

3.3 | Study 3: Hyperparameter search algorithms

Based on Studies 1 and 2, we tuned `mtry` and `min.node.size`, minimising logarithmic loss. The results of primary outcomes for each search algorithm are presented in Table 5. Effects on predictive performance did not meaningfully vary between different values for number of predictors, event fraction, or sample size. Simulation scenario-specific summary statistics are presented in Table S8 in the supplementary materials. We illustrate results specific to the scenario where $p = 16$, $EF = 0.5$, and $n = 1N$ in Figure 7. Calibration plots for all scenarios where $EF = 0.5$ are presented in Figure 8. Calibration plots for other scenarios can be found in the supplementary materials (Figures S5 and S6).

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 (Figure 7, Table 5). Random search seems to have a slightly stronger tendency to underestimate risk than grid search and model-based optimisation (Figure 8). Average discriminative performance was equal across hyperparameter search algorithms with an AUC of 0.70. Cohen's Kappa showed some variation within hyperparameter search algorithms (Figure 7). Brier score, logarithmic loss, and classification accuracy did not show variations (Figure 7).

Effects of hyperparameter search algorithms on computational time differed between simulation scenarios: while model-based optimisation had comparable runtimes across all simulation scenarios at an average of 70.50 seconds (SD = 32.06), runtimes for grid search and random search varied between simulation scenarios (Figure 6). Grid search had a mean runtime ranging from 38.01 seconds (SD = 5.97) when $p = 8, EF = 0.3, n = 0.5N$ to 18.96 minutes (SD = 2.50) when $p = 16, EF = 0.1, n = 1N$. Random search had a mean runtime ranging from 15.39 seconds (SD = 2.08) when $p = 8, EF = 0.5, n = 0.5N$ to 7.95 minutes (SD = 1.22) when $p = 16, EF = 0.1, n = 1N$. Computational time for grid search was always longer than for 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 $p = 8, EF = 0.5, n = 1N$). Doubling the sample size and holding all else constant increased computational time by a factor of between 1.88 ($p = 8, EF = 0.3$) and 2.54 ($p = 16, EF = 0.1$) in grid search, a factor of between 1.49 ($p = 8, EF = 0.3$) and 2. 41 ($p = 16, EF = 0.1$) in random search, and a factor of between 1.23 ($p = 8, EF = 0.3$) and 1.73 ($p = 16, EF = 0.3$) in model-based optimisation (Table 6). Model-based optimisation thus appeared to be less sensitive to sample size than grid search or random search.

4 | DISCUSSION

We studied the impact of the choice of hyperparameters being tuned, the optimisation criterion used, and the hyperparameter search algorithm on model performance and computational time. We conducted a simulation study for each of these three choices, using twelve data simulation scenarios varying sample size, event fraction, and number of predictors. Following the first study, we concluded that the choice of hyperparameter had small effects on discriminative performance and large effects on calibration performance and computational time. Tuning larger combinations of hyperparameters increased computational time and was associated with a decrease in calibration performance. We selected `mtry` and `min.node.size` as the optimal hyperparameter combination when considering predictive performance and computational time. Following the second study, we concluded that most optimisation criteria led to comparable performances on all outcomes. Using classification accuracy and Cohen's Kappa as optimisation criteria led to worse calibration performances. There was no noticeable effect of optimisation criterion on computational time. 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. In our setup, grid search was on average 2.76 times slower than random search. The relative length of grid search compared to model-based optimisation varied between simulation scenarios. In all but one scenario, grid search was slower than model-based optimisation. Random search was shorter than or comparable to model-based optimisation. Model-based optimisation had a computational time of around a minute in most scenarios and its runtime was less impacted by sample size than grid search and random search. We encourage the use of model-based optimisation via `tuneRanger`[8] as this is a well-performing algorithm requiring the user to make fewer choices.

Our findings that tuning `mtry` and `min.node.size` improves model performance are in line with those of Probst and colleagues[6]. Specifically, we showed this is also the case for calibration performance. Additionally, we found that model performance is not improved further by tuning larger sets of hyperparameters. These findings have important implications as they suggest that tuning is not always beneficial and may instead harm model predictive performance if many hyperparameters are tuned. We highlight that this also comes at the cost of increased computational time. It is worth noting that our study also included categorical hyperparameters, such as `splitrule`. When included in the tuning procedure, we found that `splitrule` led to a decline in calibration performance but a slight improvement in discriminative performance. It is possible specific splitting criteria favour certain aspects of model performance at the expense of others. It would be interesting to further investigate the effect of specific values for categorical hyperparameters, rather than solely their inclusion or exclusion in the tuning procedure.

We also provide previously lacking comparisons between optimisation criteria. We show using optimisation criteria based on predicted class (such as classification accuracy or Cohen's Kappa) rather than predicted risk (such as AUC, Brier score or logarithmic loss) is associated with a decrease in calibration performance. This aligns with the conceptual definition of calibration, which emphasises the agreement between predicted risks and observed proportions[11]. Given the crucial role of risk predictions in clinical decision making[34,35], we argue that the choice of optimisation criterion when tuning should be considered to avoid compromising calibration performance, and should be transparently reported.

On average, grid search and random search were slower than model-based optimisation. All three search algorithms had similar predictive performance. This is in line with Yang and Shami's findings[15] which showed grid search to needlessly increase computational time compared to other search algorithms as it did not yield a better classification accuracy than other search algorithms. We show these findings hold when considering other clinically important performance metrics, such as calibration and discrimination. However, we also found computational time is not inherently larger in certain hyperparameter search algorithms compared to others. Grid size or the number of iterations performed play a role in this. Additionally, we shed light on the impact of sample size, which does not affect all algorithms in the same way: computational time of grid search often more than doubled when sample size doubled, while it rarely increased by more than 50% in model-based optimisation. Model-based optimisation thus appears as a generally more computationally efficient algorithm, and a more reliable choice to avoid unnecessarily large computational times. Our findings provide further support to those of Probst and colleagues[8], which introduced `tuneRanger` as an efficient, well-performing, and easy to use, implementation of model-based optimisation for random forest hyperparameters in R.

This study has some limitations. First, the generalisability of our findings may be limited due to simulating all data under logistic regression. Tree-based methods are popular in part due to their flexibility[36], such that they are appropriate for a wide variety of data. Data-generating mechanisms may have a notable impact on relevant tuning procedures, as certain datasets may benefit much more from tuning than others. As our results fit well with research conducted on empirical datasets[6,15], we consider it unlikely for our choice of data-generating mechanism to have biased the results in such a way. This could be examined more

systematically in further research. Secondly, it is important to keep in mind that as our studies incorporated previous results, this may have introduced some bias in our conclusions. There may also be some bias due to the included hyperparameter combinations. `min.node.size` was selected to be in all considered combinations due to software constraints imposed by `caret`. However, our conclusions suggest `tuneRanger` to be a better, simpler, hyperparameter tuning package for random forests. This package supports tuning for `sample.fraction`. As previous findings[6] show tuning `mtry` and `sample.fraction` to lead to the most gain in discriminative performance, it would be highly relevant to compare this combination's predictive performance to that of `mtry` and `min.node.size`. Another interesting direction for further research would be to investigate interactions between hyperparameters that are tuned, optimisation criteria, and hyperparameter search algorithms.

4.1 | Conclusion

Random forests are widely used in clinical research[3,4], yet have been observed to suffer from miscalibration[12,13]. We examined the impact of various tuning procedures on random forests' predictive performance for dichotomous risk predictions and on computational time. We found the choice of tuning procedure had a large impact on predictive performance, particularly in terms of calibration, and on computational time. Specifically, tuning larger combinations of hyperparameters was detrimental to predictive performance and computational time. Optimisation criteria using predicted class rather than predicted risk had worse calibration performance than those using predicted risk. Hyperparameter search algorithms performed similarly in terms of predictive performance, but model-based optimisation was associated with faster computational times, particularly as sample size increased. This highlights the need for careful consideration of hyperparameter tuning strategies in practice.

APPENDIX

A TRUE EFFECT COMPUTATION

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 ($n = 100,000$)[37]. The first $p/2$ main effects were set to be equal (i.e., $\beta_1^{(k)} = \beta_2^{(k)} = ... = \beta_{p/2}^{(k)}$). The next $p/4$ main effects were set to be twice the strength of the first $p/2$ main effects (i.e., $\beta_{p/2+1}^{(k)} = \beta_{p/2+2}^{(k)} = ... = \beta_{3p/4}^{(k)} = 2\beta_1^{(k)}$). The final $p/4$ main effects were set to 0. All interaction effects $\boldsymbol{\gamma}^{(k)}$ were set to be equal (i.e., $\gamma_1^{(k)} = \gamma_2^{(k)} = ... = \gamma_{0.25p}^{(k)}$). 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 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 A1).

References

1. Mitchell TM. *Machine learning*. 1. McGraw-hill New York . 2007.

2. 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.

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, Van Smeden M, et al. Systematic Review Identifies the Design and Methodological Conduct of Studies on Machine Learning-Based Prediction Models. *Journal of Clinical Epidemiology* 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. *arXiv preprint arXiv:1812.02207* 2018. doi: 10.48550/arXiv.1812.02207

6. 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.

7. Bernard S, Heutte L, Adam S. Influence of hyperparameters on random forest accuracy. In: Springer. ; 2009: 171–180.

8. 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

9. Van Smeden M, Reitsma JB, Riley RD, Collins GS, Moons KG. Clinical Prediction Models: Diagnosis versus Prognosis. *Journal of clinical epidemiology* 2021; 132: 142–145. doi: 10.1016/j.jclinepi.2021.01.009

10. Wynants L, Van Smeden M, McLernon DJ, et al. Three Myths about Risk Thresholds for Prediction Models. *BMC Medicine* 2019; 17(1): 192. doi: 10.1186/s12916-019-1425-3

11. Van Calster B, McLernon DJ, Van Smeden M, et al. Calibration: the Achilles heel of predictive analytics. *BMC Medicine* 2019; 17(1): 230.

12. Benedetto U, Sinha S, Lyon M, et al. Can Machine Learning Improve Mortality Prediction Following Cardiac Surgery?. *European Journal of Cardio-Thoracic Surgery* 2020; 58(6): 1130–1136. doi: 10.1093/ejcts/ezaa229

13. 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. *Blood* 2019; 134: 5799. doi: 10.1182/blood-2019-130764

14. Strubell E, Ganesh A, McCallum A. Energy and Policy Considerations for Deep Learning in NLP. *arXiv preprint arXiv:1906.02243* 2019. doi: 10.48550/arXiv.1906.02243

15. 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

16. 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

17. 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

18. Bischl B, Casalicchio G, Feurer M, et al. OpenML Benchmarking Suites. *arXiv preprint arXiv:1708.03731* 2021. doi: 10.48550/arXiv.1708.03731

19. 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

20. 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

21. Kuhn M. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software* 2008; 28(5): 1–26. doi: 10.18637/jss.v028.i05

22. Probst P, Boulesteix AL. To Tune or Not to Tune the Number of Trees in Random Forest. *Journal of Machine Learning Research* 2017; 18(1): 6673–6690.

23. Luijken K, Groenwold RHH, Van Calster B, Steyerberg EW, Van Smeden M. 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. doi: 10.1002/sim.8183

24. Van Calster B, Van Smeden M, 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

25. Venables WN, Ripley BD. *Modern Applied Statistics with S*. New York: Springer. fourth ed. 2002. ISBN 0-387-95457-0.

26. Wickham H, Francois R, Henry L, Muller K, Vaughan D. dplyr: A Grammar of Data Manipulation. 2023. R package version 1.1.1.

27. Robin X, Turck N, Hainard A, et al. pROC: an open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics* 2011; 12: 77.

28. Revelle WR. psych: Procedures for Psychological, Psychometric, and Personality Research. 2023. R package version 2.3.3.

29. R Core Team R. R: A Language and Environment for Statistical Computing. 2023. version 4.2.3.

30. Wickham H, Averick M, Bryan J, et al. Welcome to the tidyverse. *Journal of Open Source Software* 2019; 4(43): 1686. doi: 10.21105/joss.01686

31. Garnier S, Ross N, Rudis R, Camargo PA, Sciaini M, Scherer C. viridis - Colorblind-Friendly Color Maps for R. 2021. R package version 0.6.2doi: 10.5281/zenodo.4679424

32. Dahl DB, Scott D, Roosen C, Magnusson A, Swinton J. xtable: Export Tables to LaTeX or HTML. 2019. R package version 1.8-4.

33. Allaire J, Xie Y, Dervieux C, et al. rticles: Article Formats for R Markdown. 2022. R package version 0.24.

34. Molassiotis A, Stamataki Z, Kontopantelis E. Development and Preliminary Validation of a Risk Prediction Model for Chemotherapy-Related Nausea and Vomiting. *Supportive Care in Cancer* 2013; 21(10): 2759–2767. doi: 10.1007/s00520-013-1843-2

35. Sawhney S, Tan Z, Black C, et al. Validation of Risk Prediction Models to Inform Clinical Decisions After Acute Kidney Injury. *American Journal of Kidney Diseases* 2021; 78(1): 28–37. doi: 10.1053/j.ajkd.2020.12.008

36. Breiman L. Random Forests. *Machine Learning* 2001; 45(1): 5–32. doi: 10.1023/A:1010933404324

37. Van Smeden M, Moons KG, Groot dJA, et al. Sample size for binary logistic prediction models: Beyond events per variable criteria. *Statistical Methods in Medical Research* 2019; 28(8): 2455–2474. Publisher: SAGE Publications Ltd STMdoi: 10.1177/0962280218784726

TABLE 1 Data-generating scenarios.

p	EF	n
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

TABLE 2 Hyperparameter tuning ranges in Studies 1 and 2

Hyperparameter	Default	Range	Description
`mtry`	$\sqrt{p}$ (rounded down)	1-p	Number of predictors randomly sampled to make a split.
`min.node.size`	1	1-10	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`	TRUE	TRUE, FALSE	Whether the data used to fit a single tree is sampled with or without replacement.
`sample.fraction`	1	0.1, 0.2, ..., 0.9, 1	Proportion of the data used to fit a single tree.
`splitrule`	gini	gini, hellinger, extratrees*	Splitting criterion.

*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 3 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	0.70 (0.02)	0.89 (0.27)	0.23	1.80 (00001.40)
mtry + min.node.size	0.70 (0.02)	1.05 (0.27)	0.21	278.20 (00312.40)
mtry + min.node.size + replace	0.70 (0.02)	1.03 (0.27)	0.21	639.00 (00726.70)
mtry + min.node.size + splitrule	0.71 (0.02)	1.16 (0.34)	0.28	782.50 (00782.50)
mtry + min.node.size + sample.fraction	0.70 (0.02)	1.09 (0.29)	0.23	1776.10 (01879.40)
mtry + min.node.size + replace + splitrule	0.71 (0.02)	1.13 (0.34)	0.27	1829.60 (01856.30)
mtry + min.node.size + sample.fraction + replace	0.70 (0.02)	1.07 (0.29)	0.24	3918.80 (04206.90)
mtry + min.node.size + sample.fraction + splitrule	0.71 (0.02)	1.16 (0.33)	0.27	5023.00 (04758.20)
mtry + min.node.size + sample.fraction + replace + splitrule	0.71 (0.02)	1.13 (0.33)	0.26	11185.40 (10750.00)

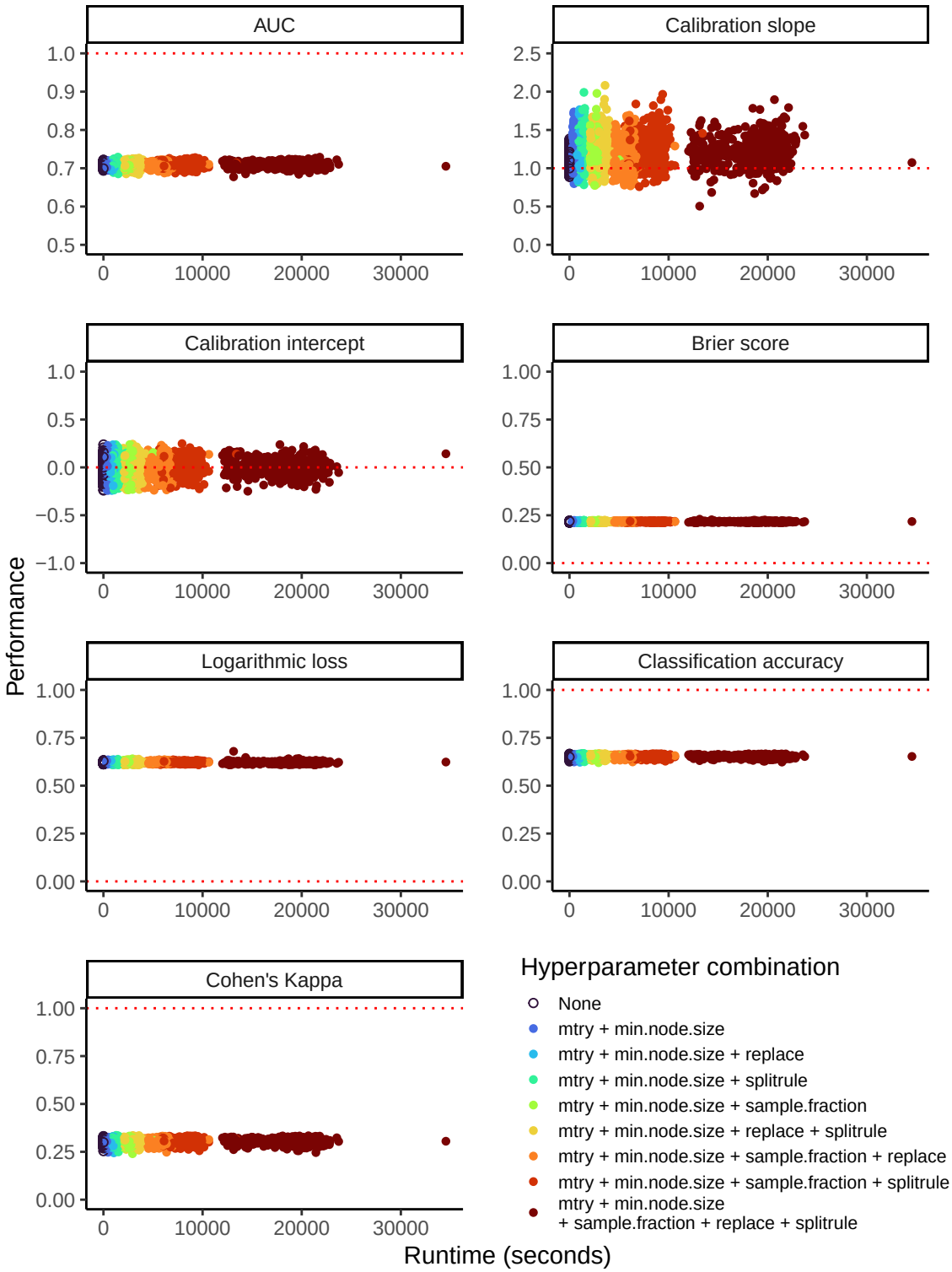

FIGURE 1 Performance of each hyperparameter tuning combination in the example scenario where $p = 16$, $EF = 0.5$, $n = 1N$. Red dotted lines show ideal performance.

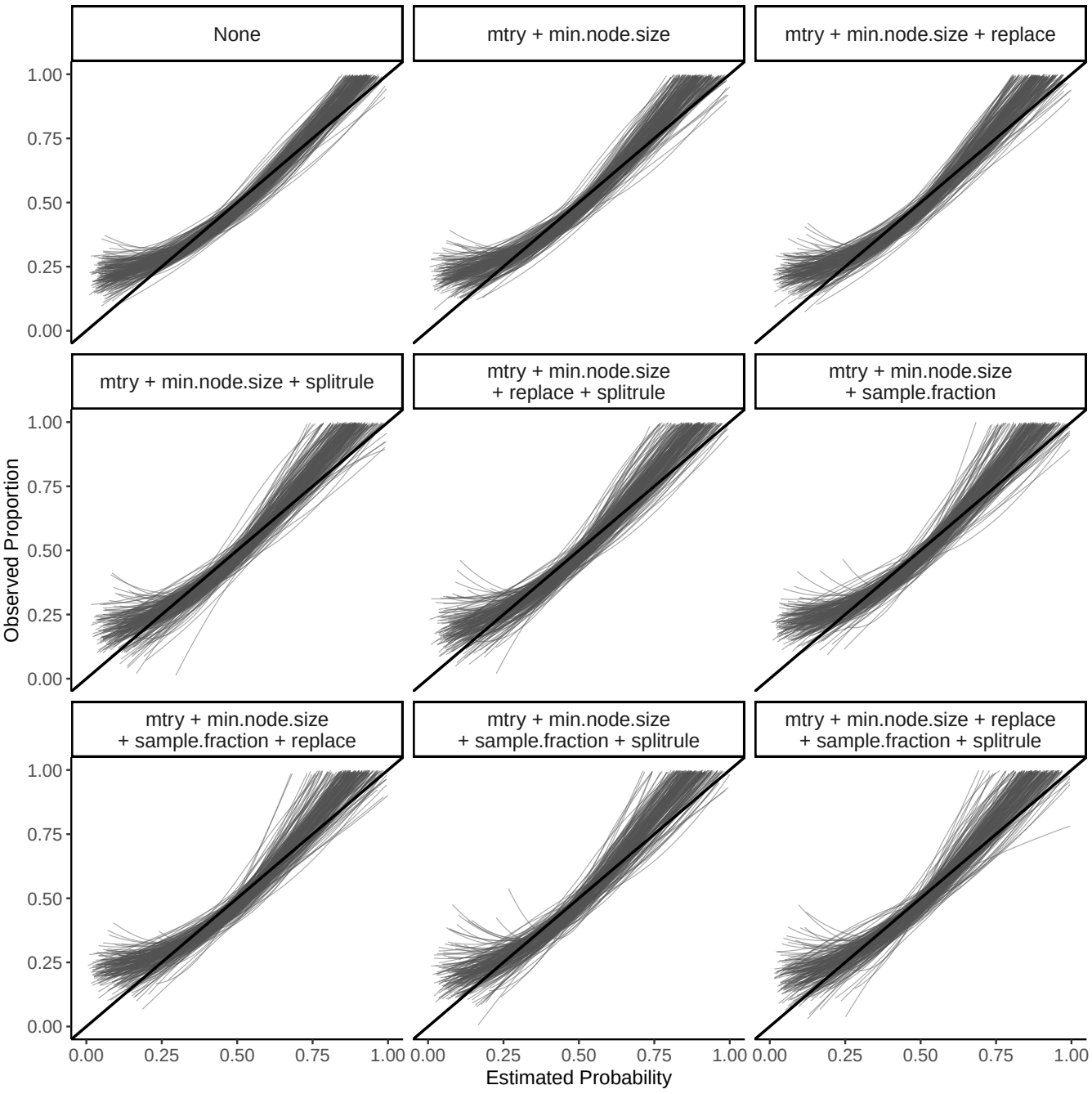

FIGURE 2 Calibration plots comparing hyperparameter combinations for every tenth dataset in scenarios where $EF = 0.5$.

TABLE 4 Average performance of optimisation criteria (aggregated over all scenarios). Rows are sorted in ascending order of RMSD(slope).

Optimisation criterion	AUC	Calibration slope	RMSD(slope)	Runtime (seconds)
	Mean (SD)	Median (IQR)		Mean (SD)
Logarithmic loss	0.70 (0.02)	1.05 (0.28)	0.20	261.70 (301.40)
Calibration slope	0.70 (0.02)	0.98 (0.27)	0.21	263.00 (301.90)
Brier score	0.70 (0.02)	1.03 (0.29)	0.22	261.40 (300.80)
Calibration intercept	0.70 (0.02)	1.13 (0.31)	0.23	262.90 (301.30)
AUC	0.70 (0.02)	1.04 (0.33)	0.27	262.20 (301.60)
Classification accuracy	0.70 (0.02)	0.90 (0.33)	0.30	262.00 (301.50)
Cohen's Kappa	0.70 (0.02)	0.78 (0.36)	0.42	262.10 (301.50)

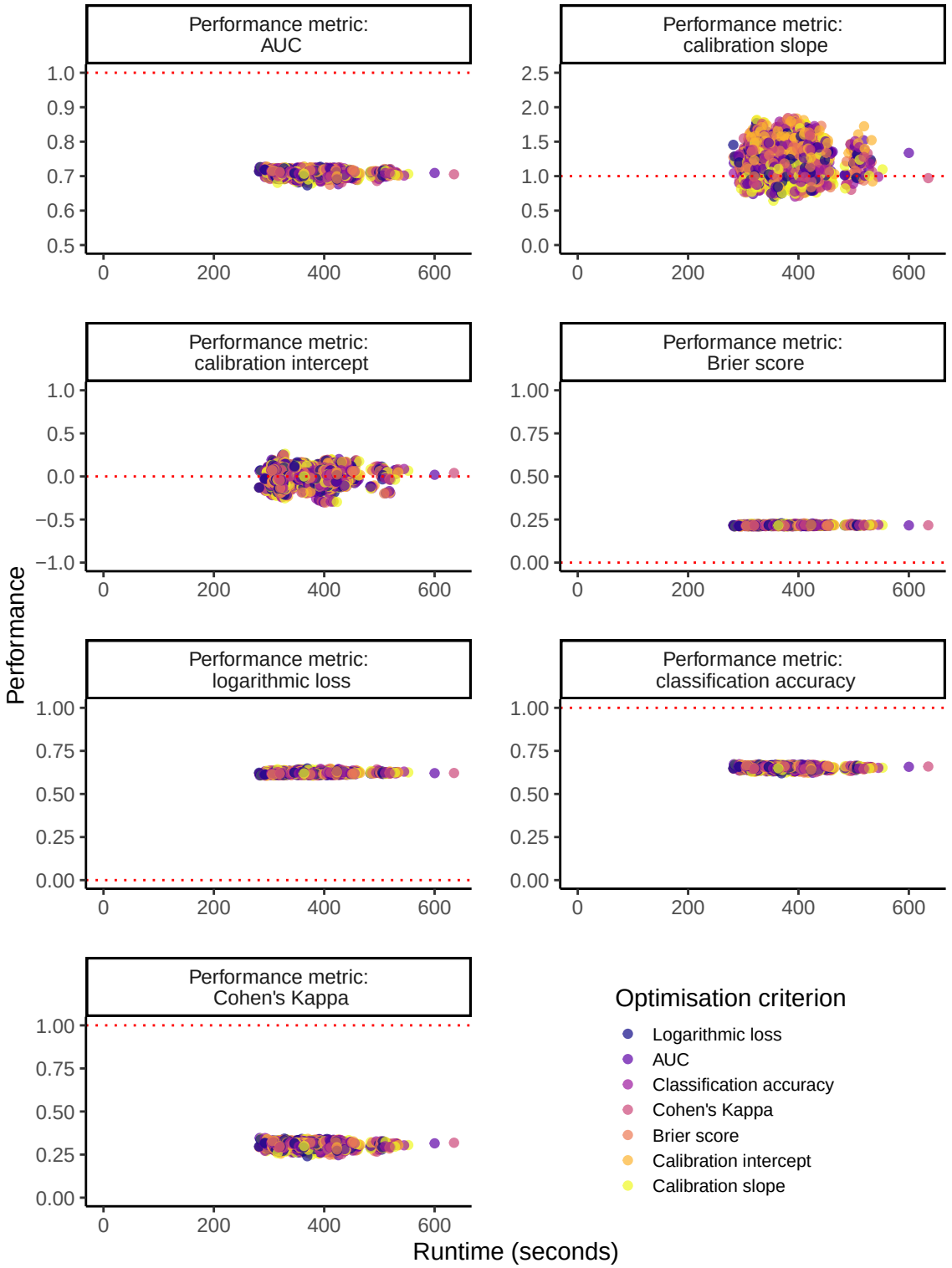

FIGURE 3 Performance of each optimisation criterion in the example scenario where $p = 16$, $EF = 0.5$, $n = 1N$. Red dotted lines show ideal performance.

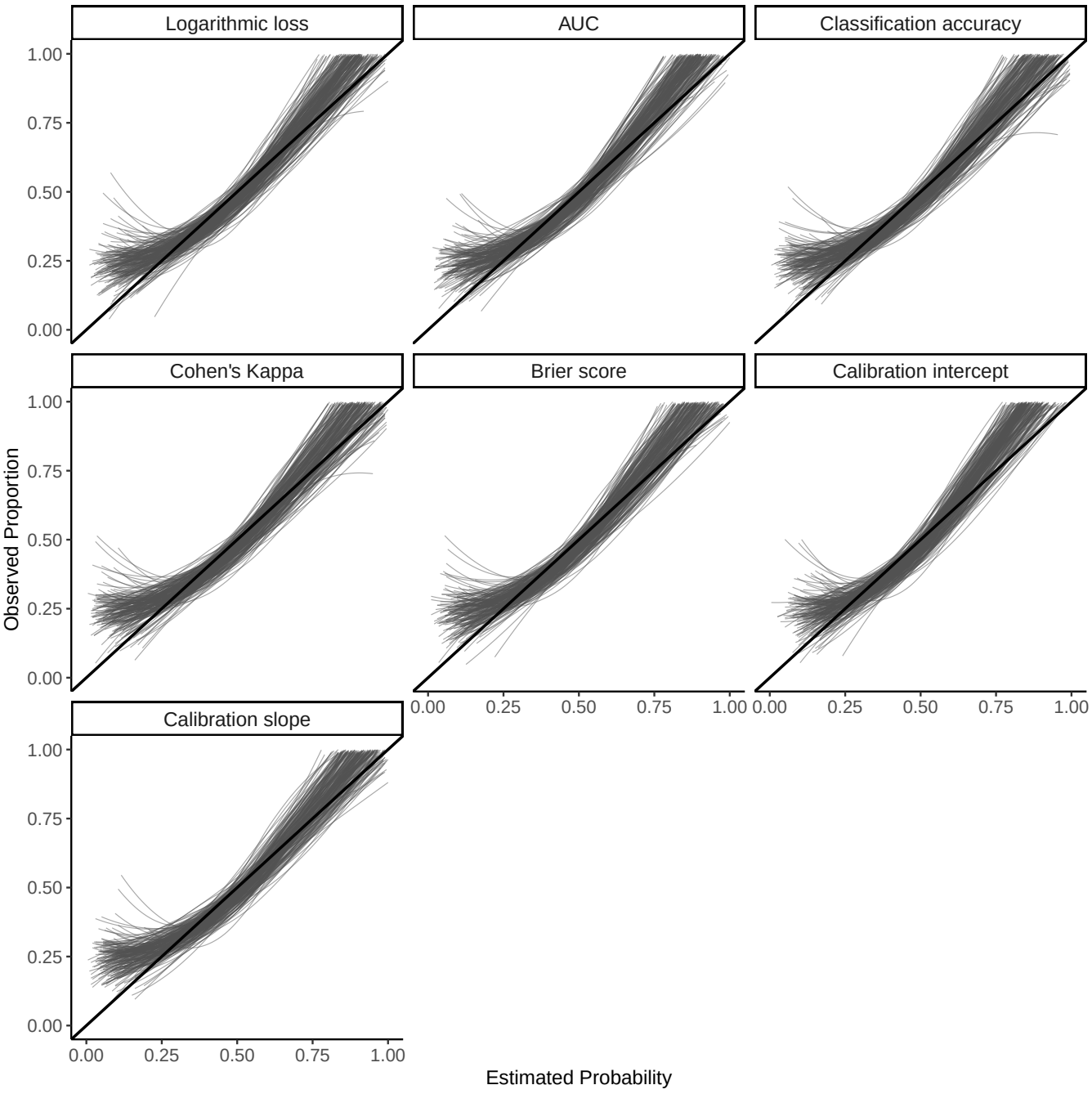

FIGURE 4 Calibration plots comparing optimisation criteria for every tenth dataset in scenarios where $EF = 0.5$.

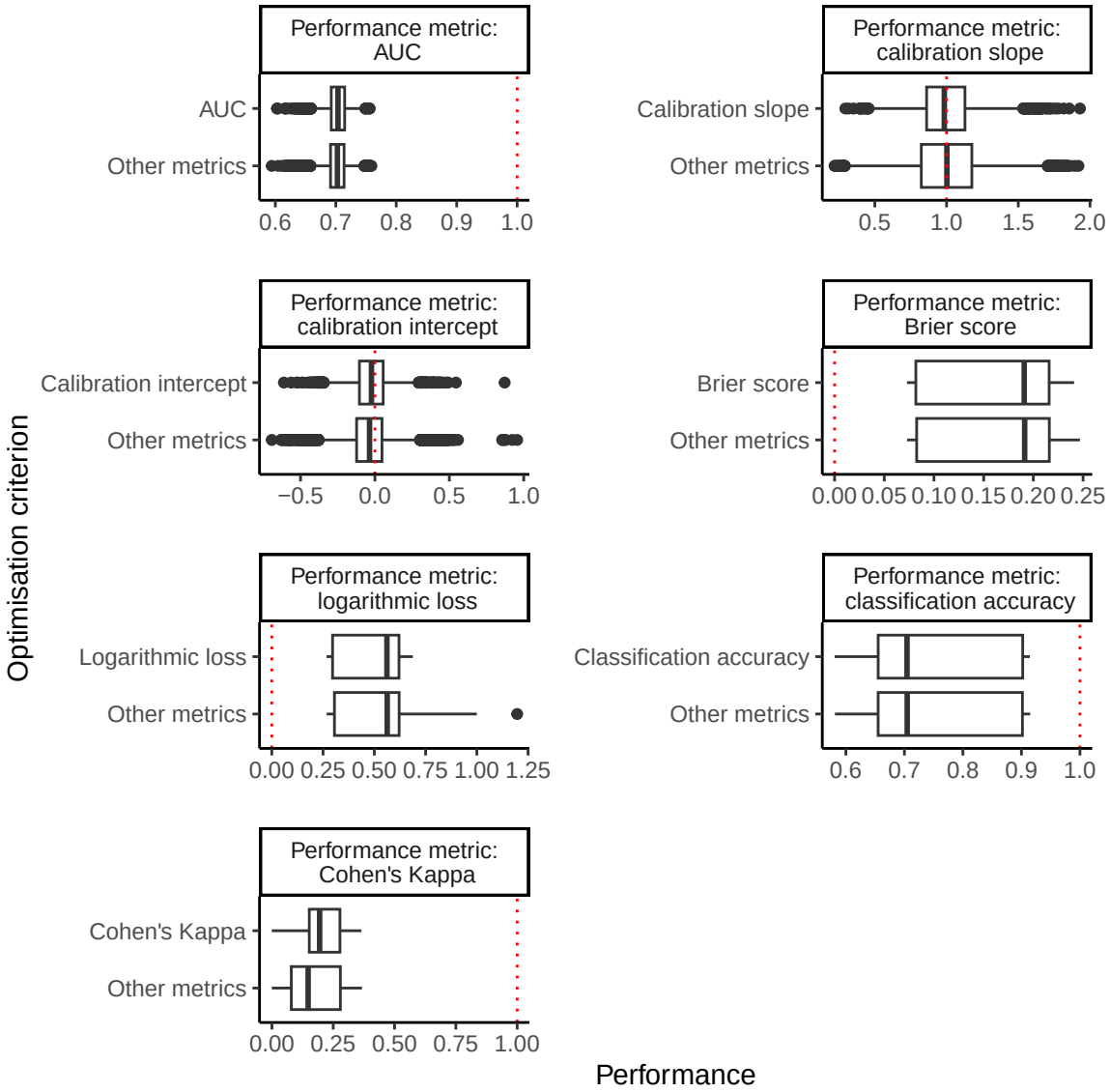

FIGURE 5 Comparison of performance where the optimisation criterion uses the performance metric to performance where the optimisation criterion does not use the performance metric. Red dotted lines show ideal performance.

TABLE 5 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	0.70 (0.02)	1.07 (0.28)	0.21	70.50 (032.06)
Random search	0.70 (0.02)	1.06 (0.29)	0.22	97.42 (127.73)
Grid search	0.70 (0.02)	1.05 (0.28)	0.21	268.28 (305.80)

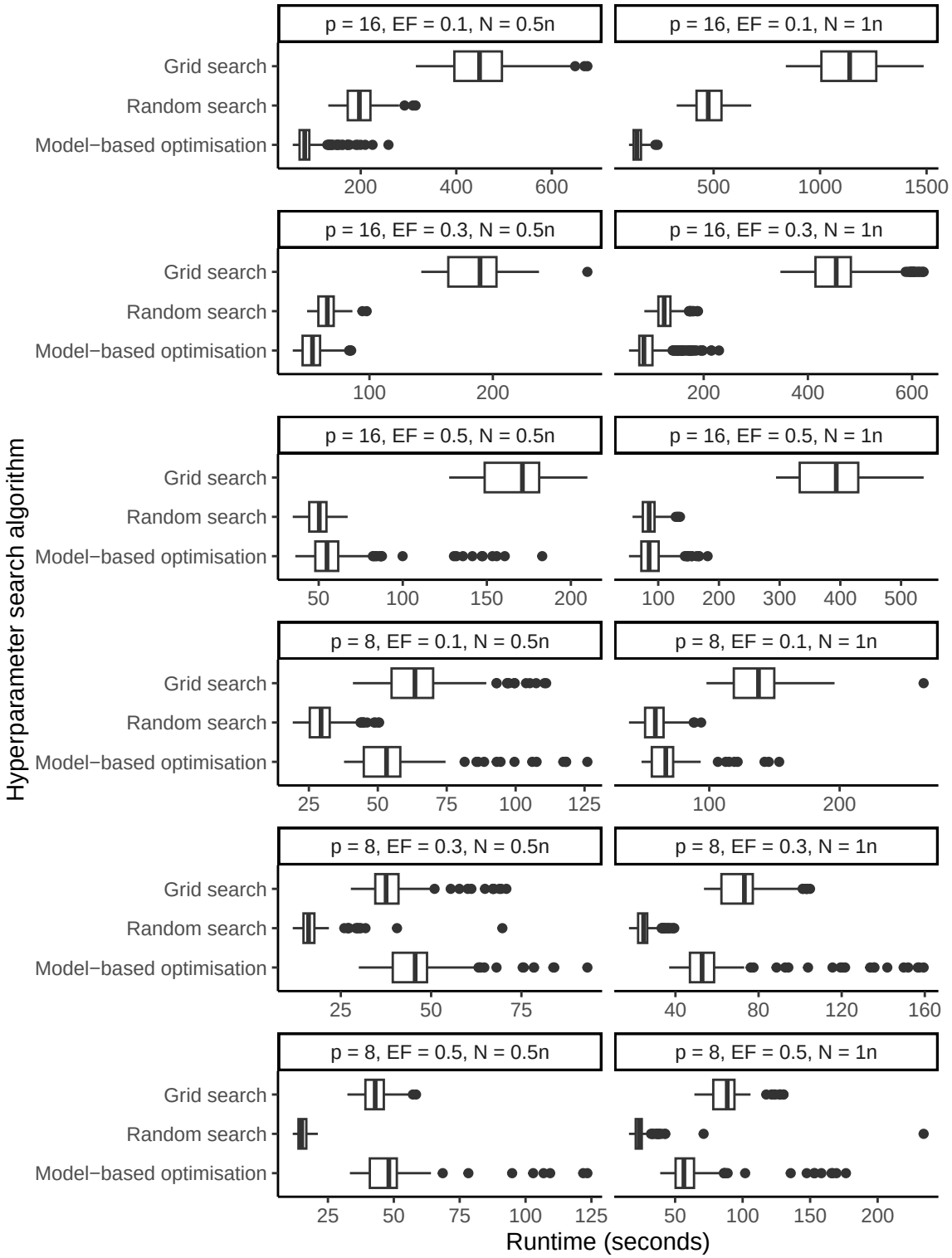

FIGURE 6 Comparison of runtimes for the hyperparameter search algorithms between scenarios.

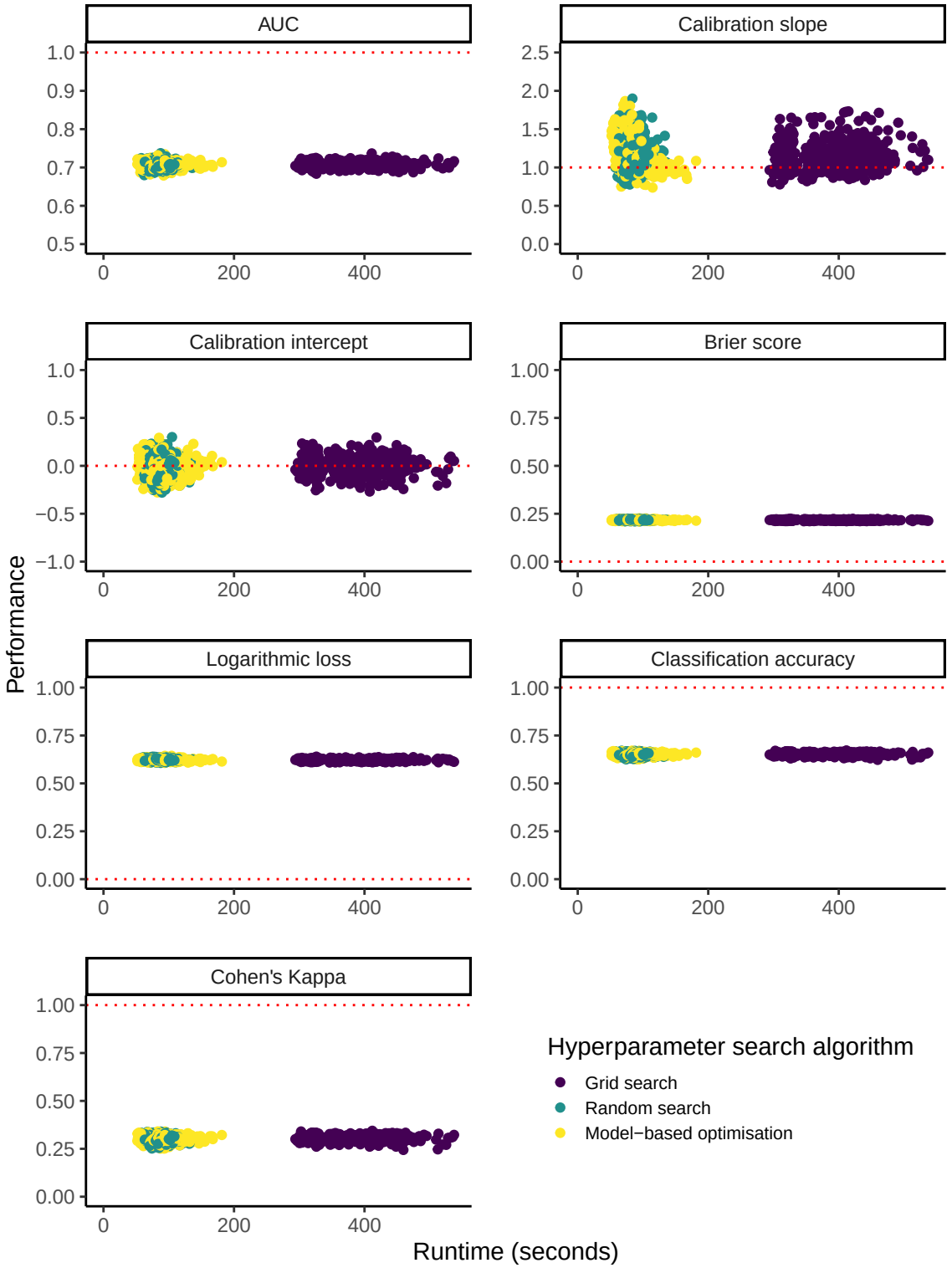

FIGURE 7 Performance of each hyperparameter search algorithm in the example scenario where $p = 16$, $EF = 0.5$, $n = 1N$. Red dotted lines show ideal performance.

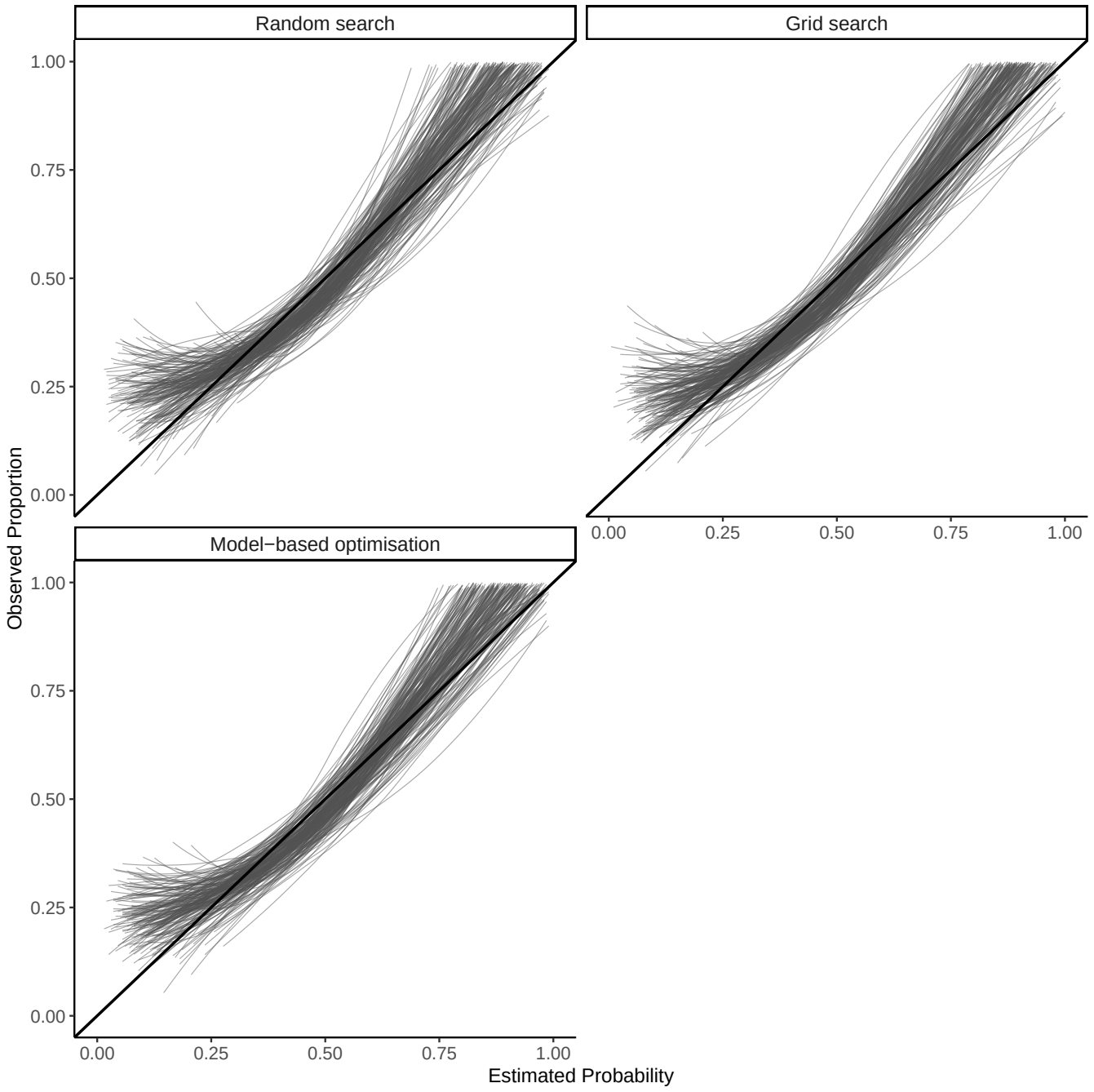

FIGURE 8 Calibration plots comparing hyperparameter search algorithms for every tenth dataset in scenarios where $EF = 0.5$.

TABLE 6 Proportional difference in runtime when doubling the sample size, holding all else constant.

Search algorithm	p	EF	Runtime ($n = 0.5N$) Mean (SD)	Runtime ($n = 1N$) Mean (SD)	Proportional difference
Grid search	8	0.10	63.80 (11.09)	136.30 (019.75)	2.14
Grid search	8	0.30	38.01 (05.97)	71.64 (009.49)	1.88
Grid search	8	0.50	43.26 (05.19)	86.61 (011.38)	2.00
Grid search	16	0.10	448.33 (66.24)	1137.30 (150.00)	2.54
Grid search	16	0.30	186.15 (21.96)	451.93 (057.52)	2.43
Grid search	16	0.50	167.45 (19.93)	388.60 (053.45)	2.32
Model-based optimisation	8	0.10	53.21 (11.51)	66.44 (012.70)	1.25
Model-based optimisation	8	0.30	44.82 (08.17)	54.92 (016.50)	1.23
Model-based optimisation	8	0.50	47.22 (09.69)	58.76 (016.73)	1.24
Model-based optimisation	16	0.10	86.04 (23.26)	142.73 (022.72)	1.66
Model-based optimisation	16	0.30	53.88 (09.08)	93.41 (028.58)	1.73
Model-based optimisation	16	0.50	56.82 (16.66)	87.69 (021.43)	1.54
Random search	8	0.10	29.43 (05.12)	58.48 (009.21)	1.99
Random search	8	0.30	16.48 (03.76)	24.63 (003.60)	1.49
Random search	8	0.50	15.39 (02.08)	23.97 (010.40)	1.56
Random search	16	0.10	198.13 (31.91)	476.86 (073.43)	2.41
Random search	16	0.30	65.38 (08.06)	125.42 (018.80)	1.92
Random search	16	0.50	49.84 (06.66)	85.02 (013.68)	1.71

TABLE A1 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