A MORE DETAILS ABOUT BLENDSEARCH

We provide detailed pseudocode of the sub-algorithms used in our framework.

Algorithm 1 `SelectThread`

Inputs: Framework-level state $\mathcal{F}$ which keeps a list of candidate search thread $\mathcal{F}.\mathbb{S}$ and a priority function $\mathcal{F}.\texttt{Priority}$.

1: $\tilde{S} = \arg\max_{S \in \mathcal{F}.\mathbb{S}} \mathcal{F}.\texttt{Priority}(S)$

2: $\tilde{S}_{\text{bak}} = \begin{cases} \arg\max_{S \in (\mathcal{F}.\mathbb{S} \setminus S_0)} \mathcal{F}.\texttt{Priority}(S) & \mathcal{F}.\mathbb{S} \setminus S_0 \neq \emptyset \\ S_0 & \mathcal{F}.\mathbb{S} \setminus S_0 = \emptyset, \end{cases}$

Outputs: $\tilde{S}, \tilde{S}_{\text{bak}}$

Algorithm 2 `BookKeeping`

Inputs: Search thread S, one entry of observation $\mathbf{x}, l, c$ generated by S, and $\mathcal{F}$.

1: $S.c \leftarrow S.c + c$

2: **if** S is a local search thread **then** Update $\{S.x_i^{\min}\}_{i \in [d]}$, and $\{S.x_i^{\max}\}_{i \in [d]}$

3: **if** $l < S.l^{\text{1st}}$ **then**

4: $\quad S.l^{\text{2nd}}, S.c^{\text{2nd}} \leftarrow S.l^{\text{1st}}, S.c^{\text{1st}}$; $S.l^{\text{1st}}, S.c^{\text{1st}} \leftarrow l, S.c$; and $S.\mathbf{x}^{best} \leftarrow \mathbf{x}$

5: $\quad$ **if** $l < \mathcal{F}.l^*$ **then** $\mathcal{F}.l^* \leftarrow l$

Algorithm 3 `InitializeSearchThread`

Inputs: Search method $\mathcal{L}$ or $\mathcal{G}$, problem P and initial point $(\mathbf{x}, l, c)$.

1: $S.\pi = \mathcal{L}.\pi$ or $\mathcal{G}.\pi$

2: Initialize bookkeeping information of S: $S.l_1 = S.l_2 = l$, $S.c^{\text{1st}} = S.c^{\text{2nd}} = S.c = c$, and $S.x_i^{\min} = S.x_i^{\max} = \mathbf{x}_i$ for $i \in P.D$

Choice of $\mathcal{L}.\Delta$ used in constructing the 'admissible' region $\mathcal{F}.\mathcal{R}$. We normalize each numeric hyperparameter into [0,1]. The setting of $\mathcal{L}.\Delta$ is decided by the local search method. For the local search method we chose, $\mathcal{L}.\Delta$ is a constant corresponding to the initial stepsize (fixed as 0.1).

Definition of `ReachableInOneStep` in Alg 5. We define local search thread S_1 to be reachable by S_2 if the distance between their incumbents is no larger than the maximal distance between the next proposal of S_2 and the incumbent of S_2. In the local search method we used, the incumbent is the currently best config, and the maximal distance is equal to the stepsize.

B MORE DETAILS ABOUT EXPERIMENTS AND ADDITIONAL RESULTS

B.1 EXPERIMENT SETUP

Settings of BO and LS. For BO, we use implementation from Optuna 2.0.0 (`https://optuna.readthedocs.io/en/stable/index.html`) with default settings for TPE sampler. For LS, we follow the implementation guidelines from Wu et al. (2021). After a local search thread is created from a particular starting point, we fix the categorical dimensions and only search for numerical dimensions in that local search thread. A local search thread S is considered to have converged (corresponding to $S.\texttt{converged}()$ in Algorithm 5) once the stepsize of the local search thread is smaller than a lower bound introduced by Wu et al. (2021).

Experiments in tuning XGBoost and LightGBM. The XGBoost and LightGBM experiments are performed in a server with Intel Xeon E5-2690 v4 2.6GHz, and 256GB RAM. A full list of hyperparameters tuned and their ranges can be found in Table 3 and Table 2. The search space for numerical hyperparameters aligns with the search space used in (Wu et al., 2021). On the same fold, the same random seed is used for LS, BO and BS. Experiments on different folds use different random seeds.

Experiments in NLP model fine tuning. For ASHA, we set min and max epochs as 1 and 16, and reduction factor 4.

Algorithm 4 `CreateNewLSCondition`

Inputs: $l, \mathcal{F}$.
Outputs: $|\mathcal{F}.\mathbb{S}| = 1$ or $l \leq \text{Median}(\{S.l^{\text{1st}}\}_{S \in \mathcal{F}.\mathbb{S} \setminus S_0})$

Algorithm 5 `DeleteAndMergeLS`

Inputs: $S, \mathcal{F}$
if S.`converged`() **then**
  $\mathcal{F}.\mathbb{S} \leftarrow \mathcal{F}.\mathbb{S} \setminus S$,
  $\mathcal{F}.\mathcal{R}.x_i^{\min} \leftarrow \mathcal{F}.\mathcal{R}.x_i^{\min} - \mathcal{L}.\Delta$, and $\mathcal{F}.\mathcal{R}.x_i^{\max} \leftarrow \mathcal{F}.\mathcal{R}.x_i^{\max} + \mathcal{L}.\Delta, \forall i \in D'$
else
  for $\forall S' \in \mathcal{F}.\mathbb{S} \setminus S$ **do**
    if $S \in S'.$`ReachableInOneStep`() & $S'.l < S.l$ **then**
      $\mathcal{F}.\mathbb{S} \leftarrow \mathcal{F}.\mathbb{S} \setminus S$
      break
    else if $S' \in S.$`ReachableInOneStep`() & $S.l < S'.l$ **then** $\mathcal{F}.\mathbb{S} \leftarrow \mathcal{F}.\mathbb{S} \setminus S'$

Result aggregation details. Aggregated rank in Figure 4(a)&(c) and 12(c) is calculated as follows: (1) per dataset per fold, the method is ranked based on the loss on validation set at each second (x-axis), starting from when there is at least one finished config evaluation in any method; (2) the rank is then averaged across datasets per fold; (3) we finally compute the average rank (line) and confidence interval (shaded area) across 10 folds. Scaled loss in Figure 4(b) is calculated similarly. Per dataset per fold, min-max scaling is applied on each method using the maximum and minimum loss along the whole performance curve across all methods.

B.2 Additional experimental results on LightGBM and XGBoost

More performance curves on LightGBM and XGBoost. The performance curves for tuning LightGBM on 3 representative datasets with an 1h budget are shown in Figure 7. We observe that the performance of LS is quite good (comparing to BO), especially on large datasets. This result is consistent with the results reported in (Wu et al., 2021), where all the hyperparameters for tuning are numerical. In our experiment of XGBoost tuning, we include categorical hyperparameters. LS performs worse in this case because the introduction of categorical hyperparameters amplifies the local search method's limitation of being trapped in local optima. The observations about `BlendSearch` for LightGBM are similar to XGBoost tuning. The performance curves on the three large datasets with a 4h budget are shown in Figure 8 and 9, where similar conclusions can be drawn.

Multi-fidelity. We compare BO and `BlendSearch` with the multi-fidelity baseline ASHA for tuning LightGBM and XGBoost in Figure 10. In this experiment, we tried two choices of fidelity dimensions with ASHA, including number of iterations and sample size (the sample size begins with 10K, so small datasets are excluded) respectively. The results show that the multi-fidelity baseline overall perform no better than BO and are significantly worse than `BlendSearch`.

Table 2: Hyperparameters tuned in LightGBM.

hyperparameter	type	range	init
tree num	int	[4, min(32768, # instance)]	4
leaf num	int	[4, min(32768, # instance)]	4
min child weight	float	[0.001, 20]	20
learning rate	float	[0.01, 0.1]	random
subsample	float	[0.6, 1.0]	random
reg alpha	float	[1e-10, 1.0]	random
reg lambda	float	[1e-10, 1.0]	random
max bin	int	[7, 1023]	random
colsample by tree	float	[0.7, 1.0]	random

Table 3: Hyperparameters tuned in XGBoost.

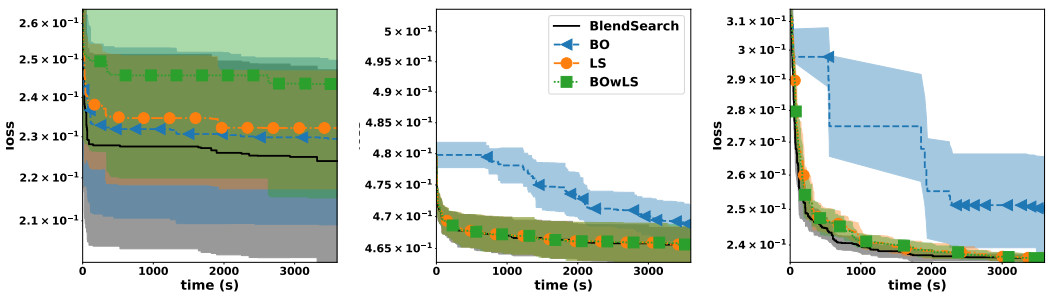
hyperparameter	type	range	init
tree num	int	[4, min(32768, # instance)]	4
leaf num	int	[4, min(32768, # instance)]	4
min child weight	float	[0.001, 20]	20
learning rate	float	[0.01, 0.1]	random
subsample	float	[0.6, 1.0]	random
reg alpha	float	[1e-10, 1.0]	random
reg lambda	float	[1e-10, 1.0]	random
colsample by level	float	[0.6, 1.0]	random
colsample by tree	float	[0.7, 1.0]	random
booster	categorical	{gbtree, gblinear}	gblinear
tree method	categorical	{auto, approx, hist}	random

Table 4: Hyperparameters tuned in DeepTables.

hyperparameter	type	range	init
early stopping rounds	int	[1, max(min(1.5M/# instance),150),10)]	10
batch size	int	[16, 1024]	512
dropout	float	[0, 0.5]	0.1
learning rate	float	[1e-4, 3e-2]	3e-4
dense dropout	float	[0, 0.5]	0.1
net	categorical	{DCN, dnn_nets}	random
auto discrete	boolean	{False, True}	random
apply gbm features	boolean	{False, True}	random
fixed embedding dim	boolean	{False, True}	random

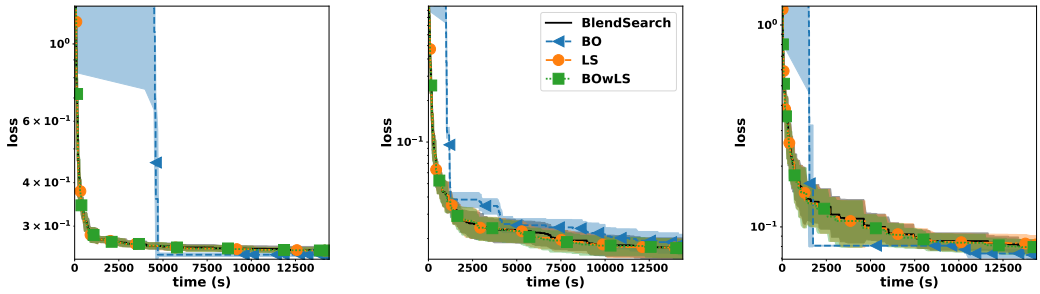
Table 5: Hyperparameters tuned in fine-tuning Turing language model.

hyperparameter	type	range	init
learning rate	float	[1e-6, 1e-3]	random
hidden dropout probability	float	[0.05, 0.4]	random
warmup proportion	float	[0.2, 0.4]	random
batch size	categorical	{16, 32}	32
epochs	int	[1, 16]	1
learning rate scheduler	categorical	{Warmup linear decay polynomial, Warmup linear, Warmup linear decay exponential}	random



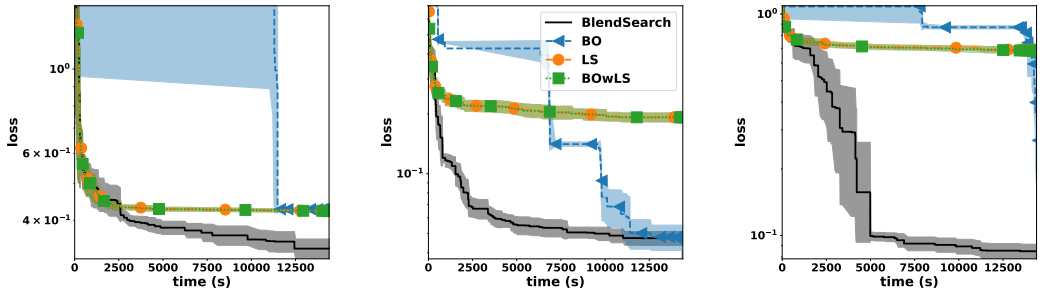
(a) blood, loss = 1 - auc (b) numerai28.6, loss = 1 - auc (c) Albert, loss = 1 - auc

Figure 7: Optimization performance curve for LightGBM (1h).



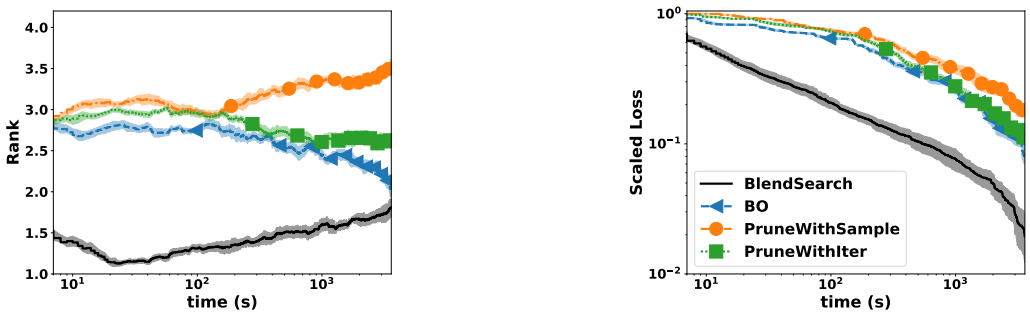
(a) Fashion-MNIST, loss = log-loss (b) dilbert, loss = log-loss (c) Covertype, loss = log-loss

Figure 8: Optimization performance curve for LightGBM (4h).



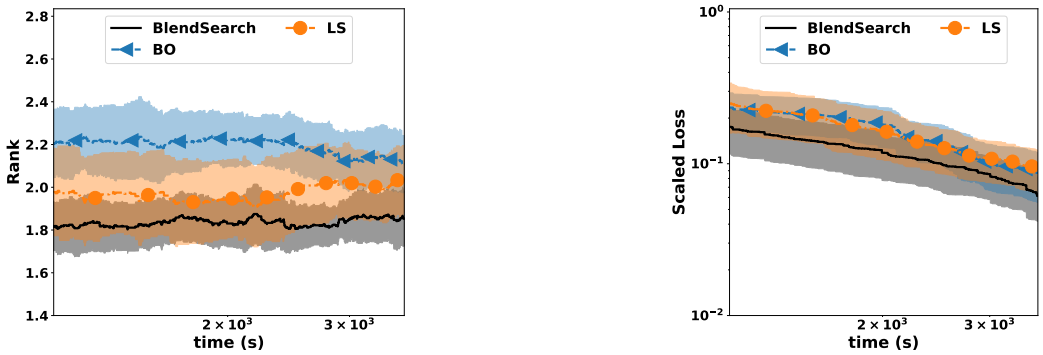
(a) Fashion-MNIST, loss = log-loss (b) dilbert, loss = log-loss (c) Covertype, loss = log-loss

Figure 9: Optimization performance curve for XGBoost (4h).



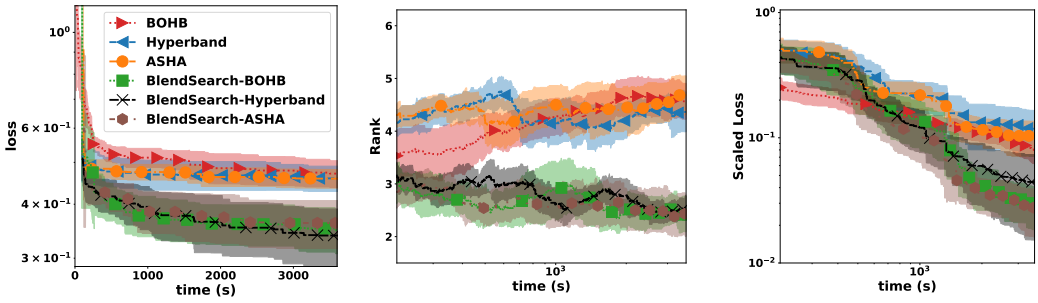
(a) Average rank (b) Scaled loss

Figure 10: Aggregated results on LightGBM and XGBoost. 'PurneWithSample' and 'PruneWithIter' represent ASHA using sample size and iteration number as resource dimension respectively. BO and `BlendSearch` are the same as those in Figure 4.



(a) Average rank (b) Scaled loss

Figure 11: Aggregated rank and scaled loss on LightGBM with random initialization.



(a) vehicle, loss = log-loss (b) Aggregated rank (c) Aggregated scaled loss

Figure 12: Performance curves on *cane*, aggregated rank and aggregated loss on DeepTables.

Ablation study on the low-cost initialization. In this work, we use low-cost initialization for the controlled dimensions of the hyperparameters. Although such information is fairly easy to obtain, we investigate our method's robustness when no controlled dimension is provided. We test `BlendSearch` in a controlled dimension agnostic setting: there are still hyperparameters with heterogeneous cost, but the controlled dimensions and a low-cost initial point is not specified as input. In such scenarios, `BlendSearch` will use random initialization and the config validator always returns 'yes'. We compare `BlendSearch` with local search and BO under such a setting using the same random initial point. In Figure 11 we report the results including the aggregated rank and scaled loss on LightGBM across half of the datasets mentioned in Section 4.1 in Figure 11(a) & (b). The results show that even if `BlendSearch` is agnostic to the controlled dimensions and a random initialization is used, it is still able to outperform both the local search method and BO.

B.3 Tuning DeepTables.

In this experiment, we tune 9-dimensional hyperparameters (5 numerical and 4 categorical as detailed in Table 4) in DeepTables. Since the training of deep neural networks are more time-consuming than that of XGBoost, we run experiments for DeepTables on the datasets where they are worse than the best known performance in the benchmark, including *'shuttle','cnae','mfeat','vehicle','phoneme', 'kc1'*. All experiments for DeepTables are performed in a server with the same CPU, 110GB RAM, and one Tesla P100 GPU. A full list of hyperparameters tuned and their ranges can be found in Table 4.

Recall that we mentioned multi-fidelity pruning strategies could be incorporated into `BlendSearch` in the config evaluator component. In this experiment, we are particularly interested showing the performance of `BlendSearch` when combined with multi-fidelity methods. To this end, we include the three state-of-the-art multi-fidelity methods, including BOHB (Falkner

et al., 2018), ASHA (Li et al., 2020), and asynchronous HyperBand (Li et al., 2017; 2020) which are shown efficient for tuning deep neural networks and the `BlendSearch` based on each of them. We use the following libraries for baselines: For BOHB, we use HpBandSter 0.7.4 (`https://github.com/automl/HpBandSter`). For ASHA and asynchronous HyperBand, we use implementations from Optuna 2.0.0. In all the methods compared, including both existing methods and variants of `BlendSearch`, the number of training epochs is used as the fidelity dimension, with maximum epochs set to be 1024, reduction factor set to be 3, and minimum epochs 4. For ASHA, we set the minimum early stopping rate to be 4 (we adopted this setting as it yields better performance comparing to the default setting, i.e., 0). The number of training epochs is used as the fidelity dimension.

`BlendSearch` incorporates existing multi-fidelity methods in the following way: Each config, either proposed by global search or local search, uses the same schedule to increase the fidelity and check its pruning condition. For example, when ASHA (Li et al., 2020), i.e., asynchronous successive halving, with a reduction factor of η, is used as the pruning strategy, after each config is evaluated by a certain fidelity, it is compared with other configs already evaluated by the same fidelity. The config will be pruned if its loss is ranked in the worst $1/\eta$. Otherwise, the fidelity is multiplied by η. In addition to the original pruning conditions specified by the multi-fidelity method, a configuration will also be pruned at a particular fidelity level where no pruning is performed yet, and the configuration does not yield superior performance (comparing to the currently-best performance) when evaluated at that fidelity level.

We present the performance of all compared methods for tuning DeepTables in Figure 12. Figure 12(a) shows the learning curves on dataset *cane* with budget 1h. Figure 12(b) and (c) show the aggregated rank and loss on all the 6 datasets within budget 1h. The performance of multi-fidelity methods are significantly improved when used in our `BlendSearch` framework.