Appendix for:

Optimizer Benchmarking Needs to Account for Hyperparameter Tuning

Prabhu Teja S [*] Florian Mai [*] Thijs Vogels Martin Jaggi François Fleuret

A. Architectures of the Models Used in Experiments

Along with the architectures examined by (Schneider et al., 2019), we experiment with an additional network and dataset. We included an additional network into our experimental setup, as DEEPOBS does not contain a word level LSTM model. Our model uses a 32-dimensional word embedding table and a single layer LSTM with memory cell size 128, the exact architecture is given in Table 5. We experiment with the IMDB sentiment classification dataset (Maas et al., 2011). The dataset contains $50,000$ movie reviews collected from movie rating website IMDB. The training set has $25,000$ reviews, each labeled as positive or negative. The rest $25,000$ form the test set. We split 20% of the training set to use as the development set. We refer the readers to DEEPOBS (Schneider et al., 2019) for the exact details of the other architectures used in this work.

Table 5: Architecture of the LSTM network used for IMDb experiments

Layer name	**Description**
Emb	[Embedding Layer; Vocabulary of 10000; Embedding dimension: 32]
LSTM_1	[LSTM; Input size: 32; Hidden dimension: 128]
FC Layer	Linear($128 \to 2$)
Classifier	Softmax(2)

B. Performance Analysis

We show the full performance plots of all variants of SGD, Adam, and Adagrad we experimented with in Figure 5.

C. How Likely Are We to Find Good Configurations?

In Figure 1 we showed the chance of finding the optimal hyperparameter setting for some of the optimizers considered, in a problem agnostic setting. Here we delve into the case where we present similar plots for each of the problems considered in Section 5.

A natural question that arises is: Given a budget K, what is the best optimizer one can pick? In other words, for a given budget what is the probability of each optimizer finding the best configuration? We answer this with a simple procedure. We repeat the runs of HPO for a budget K, and collect the optimizer that gave the best result in each of those runs. Using the classical definition of probability, we compute the required quantity. We plot the computed probability in Figure 6. It is very evident for nearly all budgets, Adam-LR is always the best option for 4 of the problems. SGD variants emerge to be better options for CIFAR-100 and Char-RNN at later stages of HPO. For some of the problems like VAEs, LSTM, it is very obvious that Adam-LR is nearly always the best choice. This further strengthens our hypothesis that adaptive gradient methods are more tunable, especially in constrained HPO budget scenarios.

D. Computing the Expected Maximum of Random Samples

The following is a constructive proof of how to compute the expected value that the bootstrap method converges to in the limit of infinite re-sampling. It is a paraphrase of Dodge et al. (2019, Section 3.1), but due to inaccuracies in Equation (1) in their paper, we repeat it here for clarity.

Let $x_1, x_2 \dots x_N \sim \mathcal{X}$ be N independently sampled values. Let the random variable $\mathbf{Y}$ be the maximum of a random subset of size S from $x_1, x_2 \dots x_N$ where $S \leq N$. For representational convenience, let them be the first S samples. So, $\mathbf{Y} = \max\{x_1, \dots, x_S\}$. We are interested in computing $\mathbb{E}[\mathbf{Y}]$. This can be computed as

$$\mathbb{E}[\mathbf{Y}] = \sum_y y \cdot P(\mathbf{Y} = y)$$

for discrete $\mathbf{Y}$, with $P(\mathbf{Y} = y)$ be the probability mass function of $\mathbf{Y}$. We can write

$$P(\mathbf{Y} = y) = P(\mathbf{Y} \leq y) - P(\mathbf{Y} < y)$$

As $x_i\ \forall i$ are iid sampled,

$$\begin{aligned} P(\mathbf{Y} \leq y) &= P(\max_{i=1\dots S} x_i \leq y) \\ &= \prod_{i=1}^{S} P(x_i \leq y) \\ &= P(X \leq y)^S \end{aligned}$$

$P(X \leq y)$ can be empirically estimated from data as the sum of normalized impulses.

$$P(X \leq y) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{x_i \leq y} \tag{1}$$

Thus,

$$\mathbb{E}[\mathbf{Y}] = \sum_y y(P(X \leq y)^S - P(X < y)^S) \tag{2}$$

A very similar equation can be derived to compute the variance too. Variance is defined as $Var(\mathbf{Y}) = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. The second operand is given by Equation (2). The first operand (for discrete distributions) can be computed as

$$\mathbb{E}[\mathbf{Y}^2] = \sum_y y^2(P(X \leq y)^S - P(X < y)^S) \tag{3}$$

Given the iterates (not incumbents) of Random Search, the expected performance at a given budget can be estimated by Equation (2) and the variance can be computed by Equation (3).

E. Aggregating the Performance of Incumbents

In Procedure 1, we propose returning all the incumbents of the HPO algorithm. Here we propose the use of an aggregation function that helps create a comparable scalar that can used in a benchmarking software like DEEPOBS to rank the performance of optimizers that takes into cognizance the ease-of-use aspect too.

E.1. Aggregating Function

For the aggregation function discussed, we propose a simple convex combination of the incumbent performances and term it ω-tunability. If $\mathcal{L}_t$ be the incumbent performance at budget t, we define ω-tunability as

$$\omega\text{-tunability} = \sum_{t=1}^{T} \omega_t \mathcal{L}_t$$

where $w_t > 0 \; \forall$ t and $\sum_t w_t = 1$

By appropriately choosing the weights $\{\omega_t\}$, we can interpolate between our two notions of tunability in Section 2. In the extreme case where we are only interested in the peak performance of the optimizer, we can set $\omega_T = 1$ and set the other weights to zero. In the opposite extreme case where we are interested in the "one-shot tunability" of the optimizer, we can set $\omega_1 = 1$. In general, we can answer the question of "How well does the optimizer perform with a budget of K runs?" by setting $\omega_i = \mathbf{1}_{\mathrm{i=K}}$. Figure 4 and Figure 5 can also be computed as $\omega_i = \mathbf{1}_{\mathrm{i=K}}$.

While the above weighting scheme is intuitive, merely computing the performance after expending HPO budget of K does not consider the performance obtained after the previous $K - 1$ iterations i.e. we would like to differentiate the cases where a requisite performance is attained by tuning an optimizer for K iterations and another for K_1 iterations, where $K_1 \gg K$. Therefore, we employ a weighting scheme as follows: By setting $\omega_i \propto (T - i)$, our first one puts more emphasis on the earlier stages of the hyperparameter tuning process. We term this weighting scheme *Cumulative Performance-Early*(*CPE*). The results of the various optimzers' *CPE* is shown in Table 6. It is very evident that Adam-LR fares the best across tasks. Even when it is not the best performing one, it is quite competitive. Thus, our observation that Adam-LR is the easiest-to-tune i.e. it doesn't guarantee the best performance, but it gives very competitive performances early in the HPO search phase, holds true.

For a benchmarking software package like DEEPOBS, we suggest the use of *CPE* to rank optimziers, as it places focus on ease-of-tuning. This supplements the existing peak performance metric reported previously.

Optimizer	FMNIST(%)$\uparrow$	CIFAR 10(%)$\uparrow$	CIFAR 100(%)$\uparrow$	IMDb(%)$\uparrow$	WRN 16(4)(%)$\uparrow$	Char-RNN(%)$\uparrow$	MNIST-VAE$\downarrow$	FMNIST-VAE$\downarrow$	Quadratic Deep$\downarrow$
Adam-LR	**91.6**	78.8	**42.0**	85.9	**95.3**	56.9	28.9	**24.3**	89.9
Adam	91.3	77.3	38.1	83.4	94.5	54.2	33.1	25.7	95.4
SGD-M^{C}W^C	90.8	81.0	38.8	78.7	**95.3**	53.9	54.0	27.9	**87.4**
SGD-MW	90.5	79.6	33.2	75.2	95.0	44.4	35.2	26.5	87.5
SGD-M^{C}D	91.1	**82.1**	39.2	80.5	95.2	49.6	54.3	29.8	87.5
Adagrad	91.3	76.6	29.8	84.4	95.0	55.6	30.7	25.9	90.6
Adam-W^{C}D	**91.6**	79.4	35.1	**86.0**	95.1	**57.4**	**28.6**	**24.3**	92.8
SGD-LR	90.4	76.9	30.6	68.1	94.7	39.9	53.4	26.2	89.3
SGD-M	90.5	77.8	39.8	73.8	94.9	50.7	37.1	26.3	88.2
SGD-M^C	90.7	78.8	**42.0**	79.0	95.0	55.5	54.1	28.5	88.1

Table 6: *CPE* for the various optimizers experimented. It is evident that Adam-LR is the most competitive across tasks.

F. Results for Computation Time Budgets

Using number of hyperparameter configuration trials as budget unit does not account for the possibility that optimizers may require different amounts of computation time to finish a trial. While the cost for one update step is approximately the same for all optimizers, some require more update steps than others before reaching convergence.

To verify that our results and conclusions are not affected by our choice of budget unit, we simulate the results we would have obtained with a computation time budget in the following way. For a given test problem (e.g., CIFAR-10), we compute the minimum number of update steps any optimizer has required to finish 100 trials, and consider this number to be the maximum computation budget. We split this budget into 100 intervals of equal size. Using the bootstrap (Tibshirani & Efron, 1993), we then simulate 1,000 HPO runs, and save the best performance achieved at each interval. Note that sometimes an optimizer can complete multiple hyperparameter trials in one interval, and sometimes a single trial may take longer than one interval. Finally, we average the results from all 1,000 HPO runs and compute the same summary across datasets as in Section 5.2.

Figure 7 shows that the conclusions do not change when using computation time as budget unit. In fact, the graphs show almost the exact same pattern as in Figure 3, where number of hyperparameter trials is the budget unit.

G. Plotting Hyperparameter Surfaces

In Section 2, we hypothesize that the performance as a function of the hyperparameter, e.g., learning rate, of an optimizer that performs well with few trials has a wider extremum compared to an optimizer that only performs well with more trials.

In Figure 8, we show a scatter plot of the loss/accuracy surfaces of SGD-M^{C}W^C and Adam-LR as a function of the learning rate, which is their only tunable hyperparameter. The plots confirm the expected behavior. On MNIST VAE, FMNIST VAE, and Tolstoi-Char-RNN, Adam-LR reaches performances close to the optimum on a wider range of learning rates than SGD-M^{C}W^C does, resulting in substantially better expected performances at small budgets ($k = 1, 4$) as opposed to SGD-M^{C}W^C, even though their extrema are relatively close to each other. On CIFAR10, the width of the maximum is similar, leading to comparable performances at low budgets. However, the maximum for SGD-M^{C}W^C is slightly higher, leading to better performance than Adam-LR at high budgets.

H. Interplay between momentum and learning rate

We ran an additional experiment using 'effective learning rate' (Shallue et al., 2019) that combines learning rate γ, and momentum μ of SGD to compute the effective learning rate γ^{eff}. Intuitively, γ^{eff}quantifies the contribution of a given minibatch to the overall training. This is defined as

$$\gamma^{\text{eff}} = \frac{\gamma}{1 - \mu}$$

We designed a variant of SGD-MW, called SGD-LR$_{\text{eff}}$, where we sampled γ and γ^{eff}independently from lognormal priors calibrated as usual, and compute the momentum(μ) as $\mu = \max(0, (1 - \frac{\gamma}{\gamma^{\text{eff}}}))$, hence accounting for interplay between learning rate and momentum. We plot the performance comparisons between SGD-MW and SGD-LR$_{\text{eff}}$ in Figure 9, and provide a plot of the aggregated relative performance in Figure 10. The results show that indeed SGD-LR$_{\text{eff}}$ improves over SGD-MW in the low-budget regime, particularly on classification tasks. We attribute this to the fact that SGD-LR$_{\text{eff}}$ is effective at exploiting historically successful (γ, μ) pairs. For large budgets, however, SGD-LR$_{\text{eff}}$ performs increasingly worse than SGD-MW, which can be explained by the fact that SGD-MW has a higher chance of exploring new configurations due to the independence assumption. Despite the improvement in low-budget regimes, SGD variants, including the new SGD-LR$_{\text{eff}}$ variant, remain substantially below Adam-LR in all budget scenarios. Hence, our conclusion remains the same.

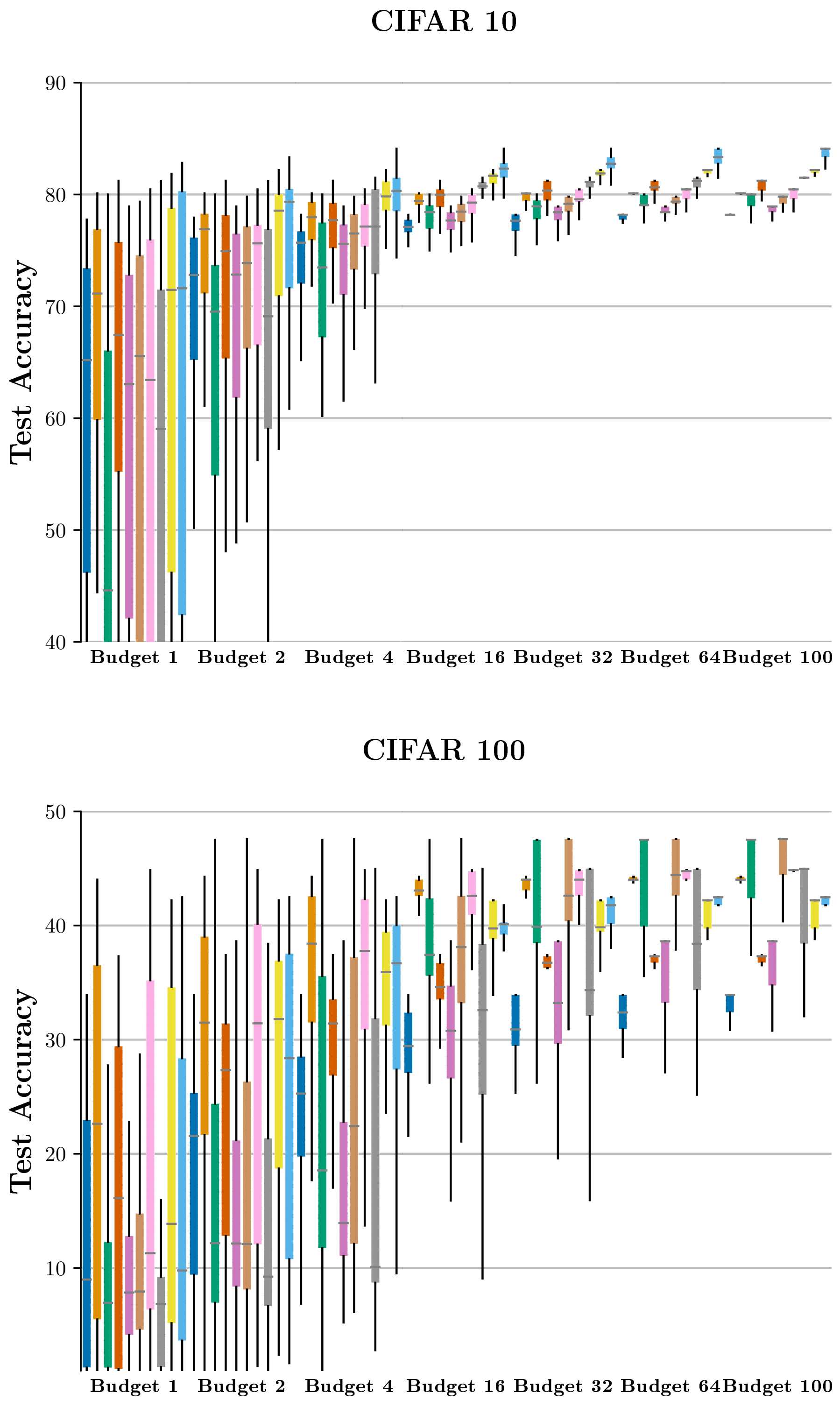
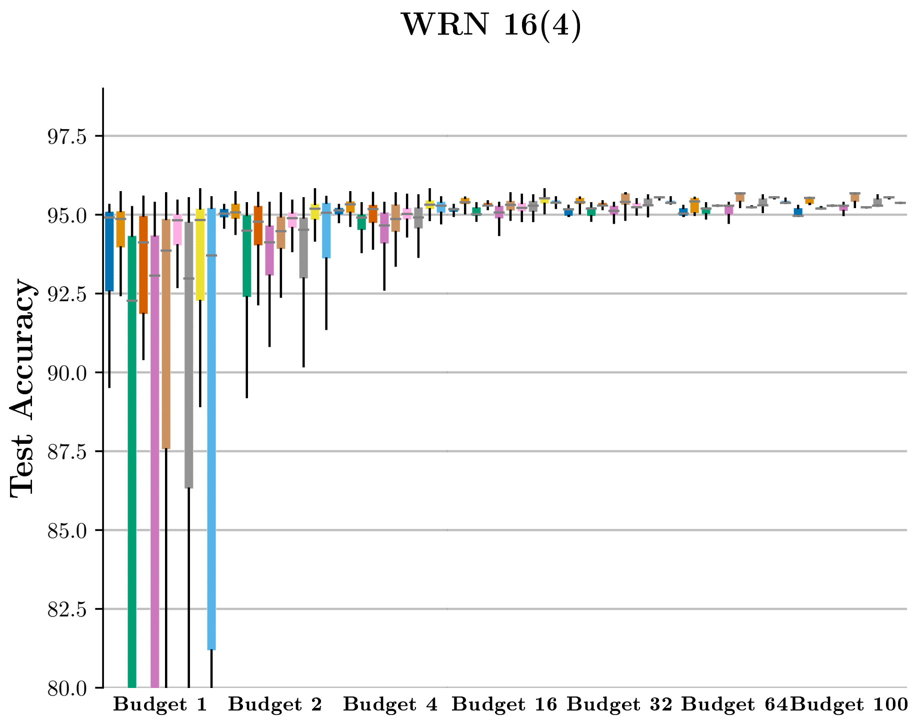


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**



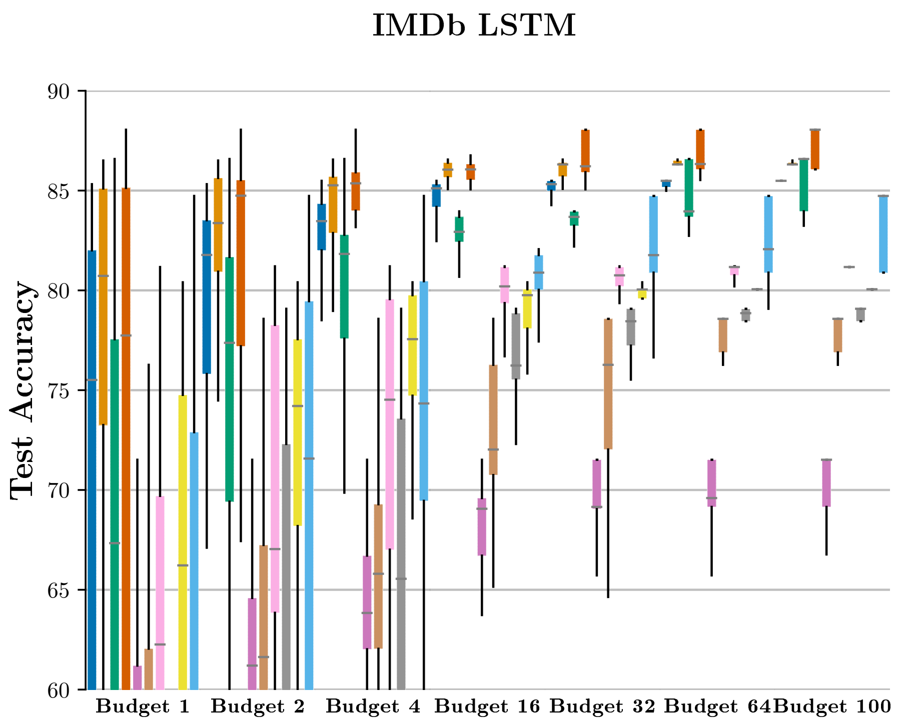
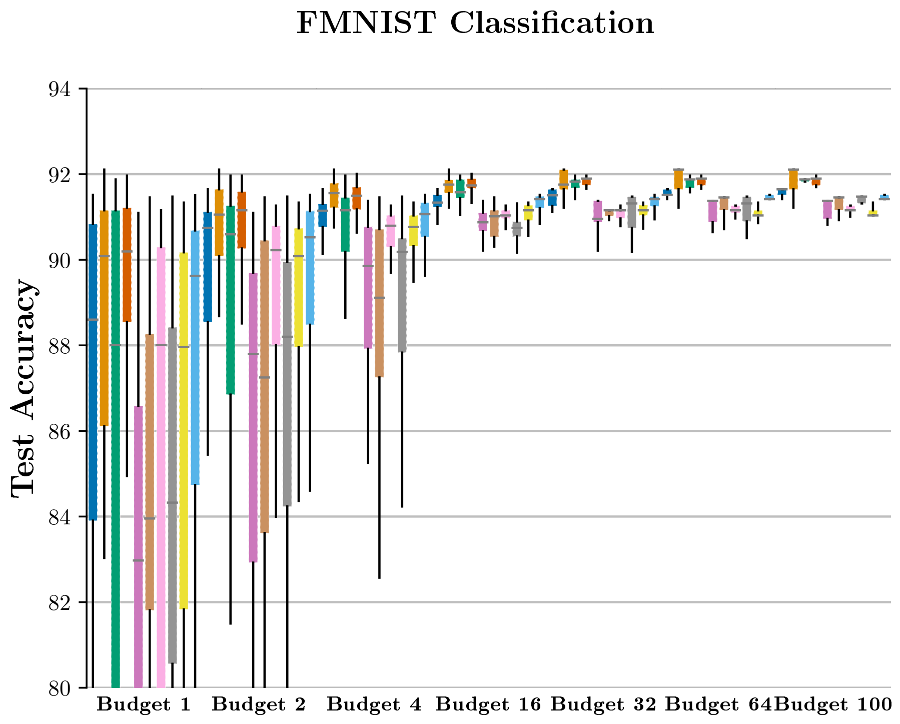


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**



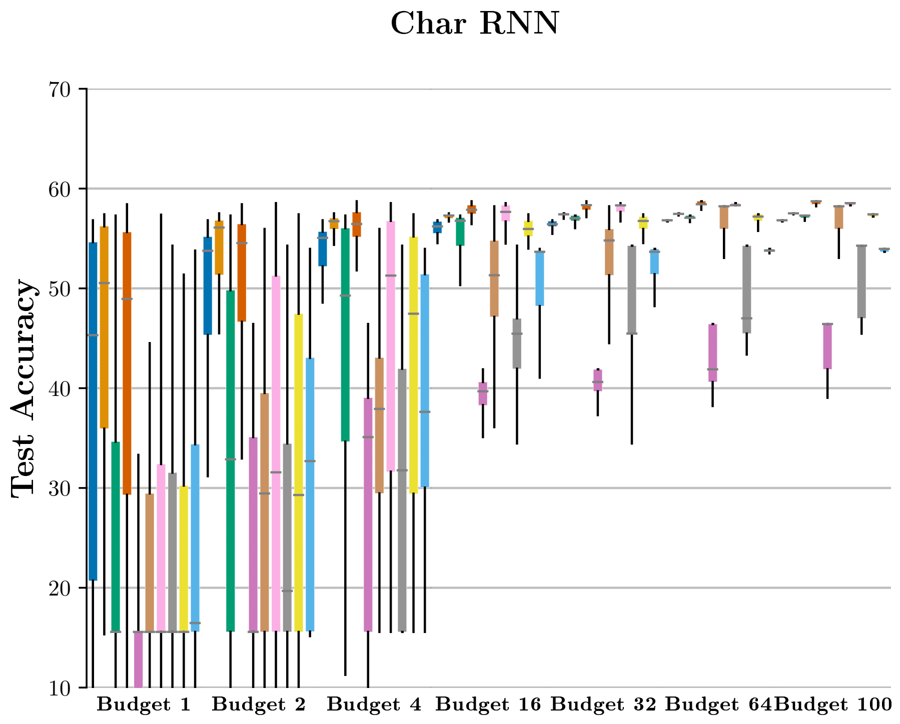
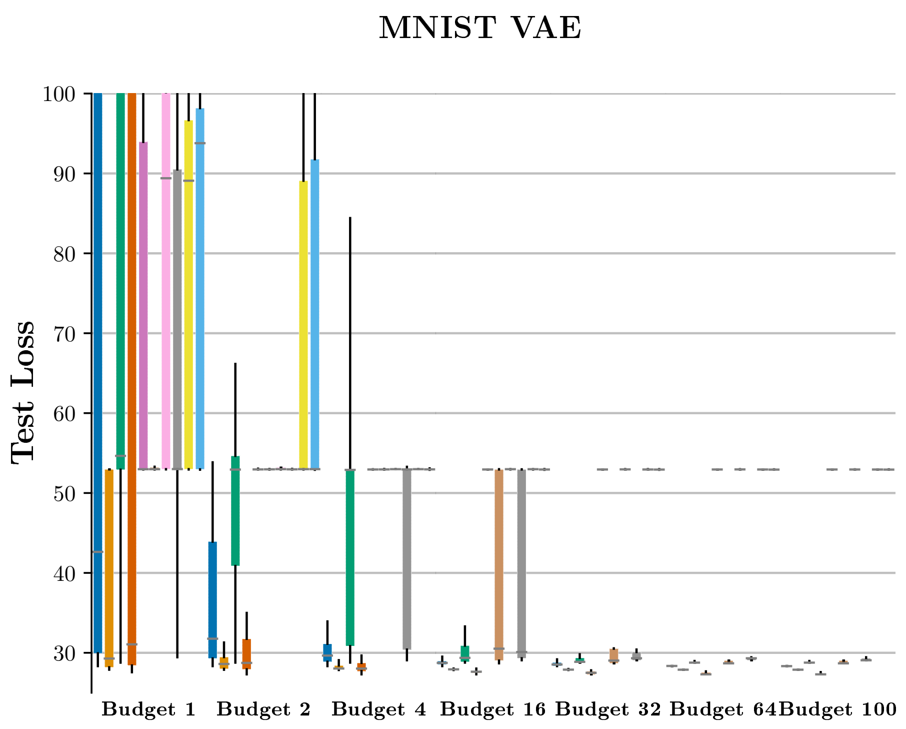


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**



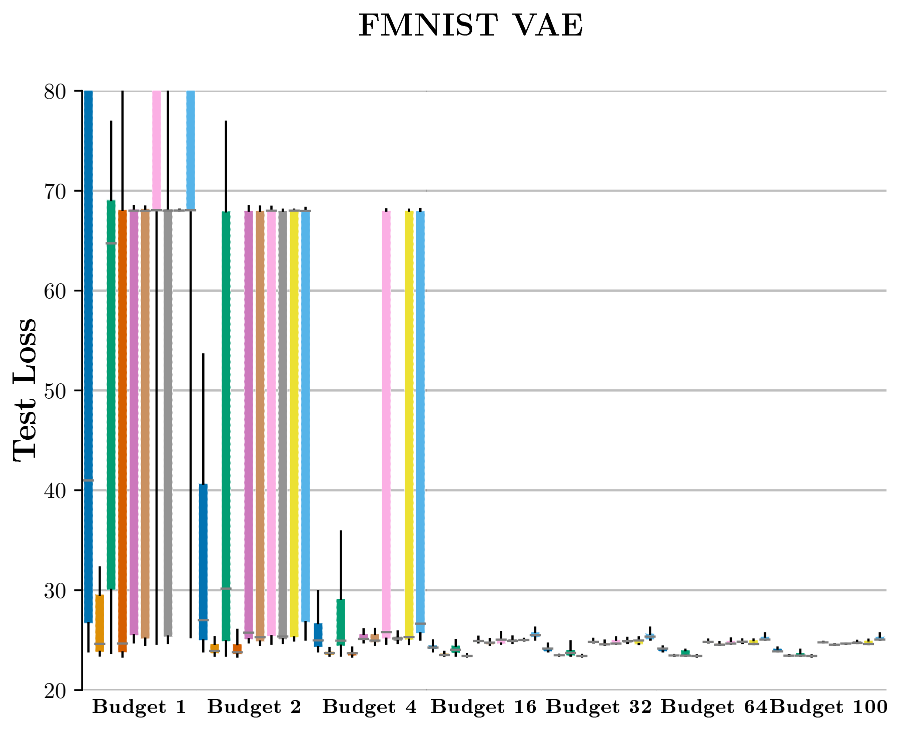


Figure 5: **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D**

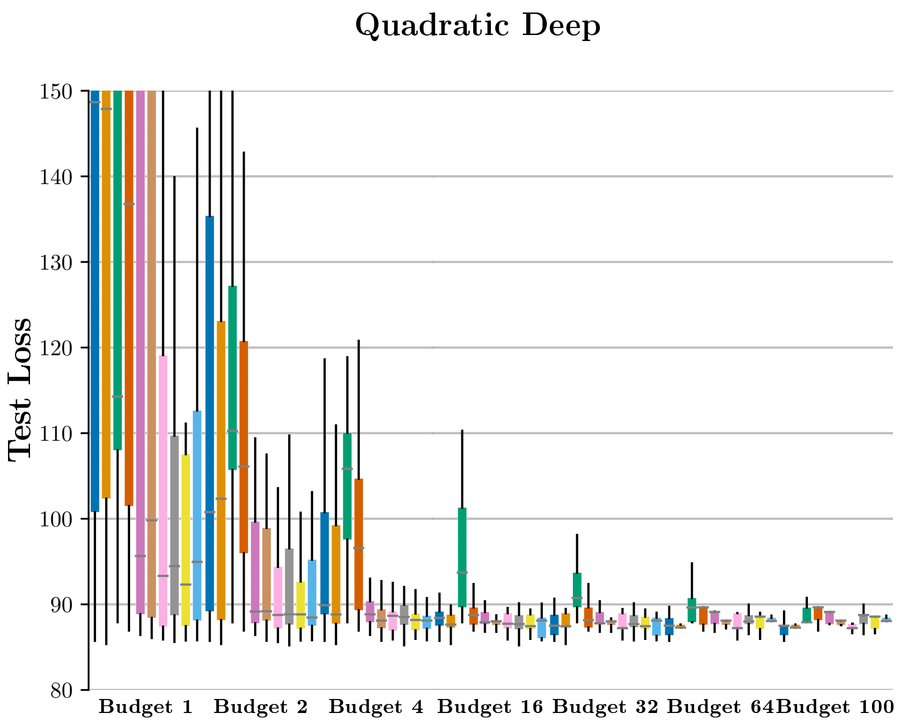


Figure 5: We show the performance of **Adagrad**, **Adam-LR**, **Adam**, **Adam-W^{C}D**, **SGD-LR**, **SGD-M**, **SGD-M^C**, **SGD-MW**, **SGD-M^{C}W^C**, and **SGD-M^{C}D** over all the experiments. We plot the on the x-axis the number of the hyperparameter configuration searches, on the y-axis the appropriate performance.

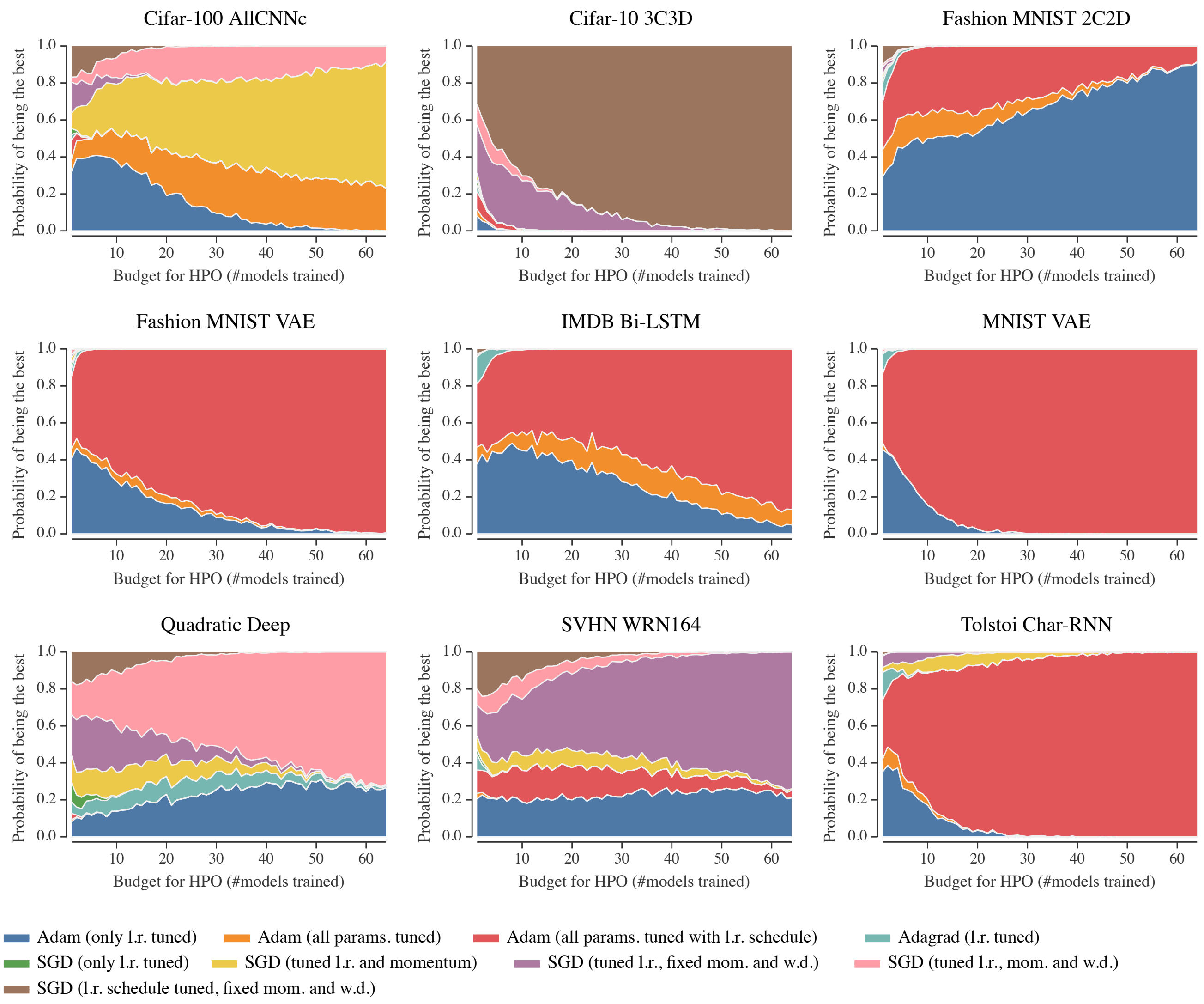


Figure 6: Which optimizer for which budget? Given a tuning budget K (x-axis), the stacked area plots above show how likely each optimizer (colored bands) is to yield the best result after K steps of hyperparameter optimization. For example, for the IMDB LSTM problem, for a small budget, Adam-LR is the best choice (with ~ 0.8 probability), whereas for a larger search budget > 50, tuning the other parameters of 'Adam' is likely to pay off.

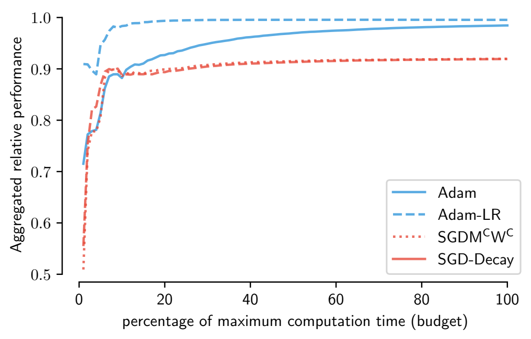


Figure 7: Aggregated relative performance of each optimizer across datasets.

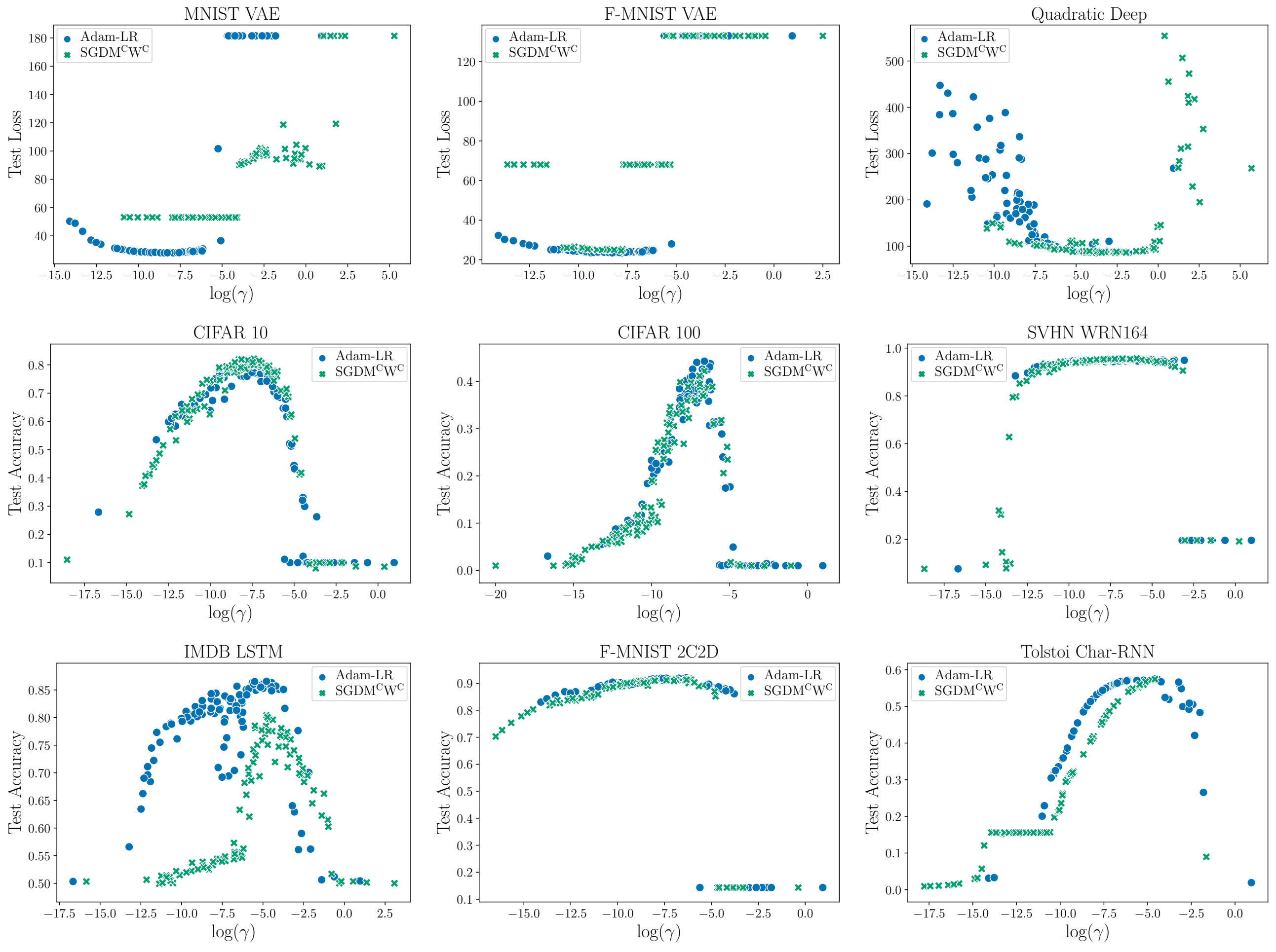


Figure 8: Scatter plot of performance of Adam-LR and SGD-M^{C}W^C by learning rate value. For better visibility, we shift the learning rate values of SGD-M^{C}W^C in such a way that the minima of both optimizers are at the same position on the x-axis.

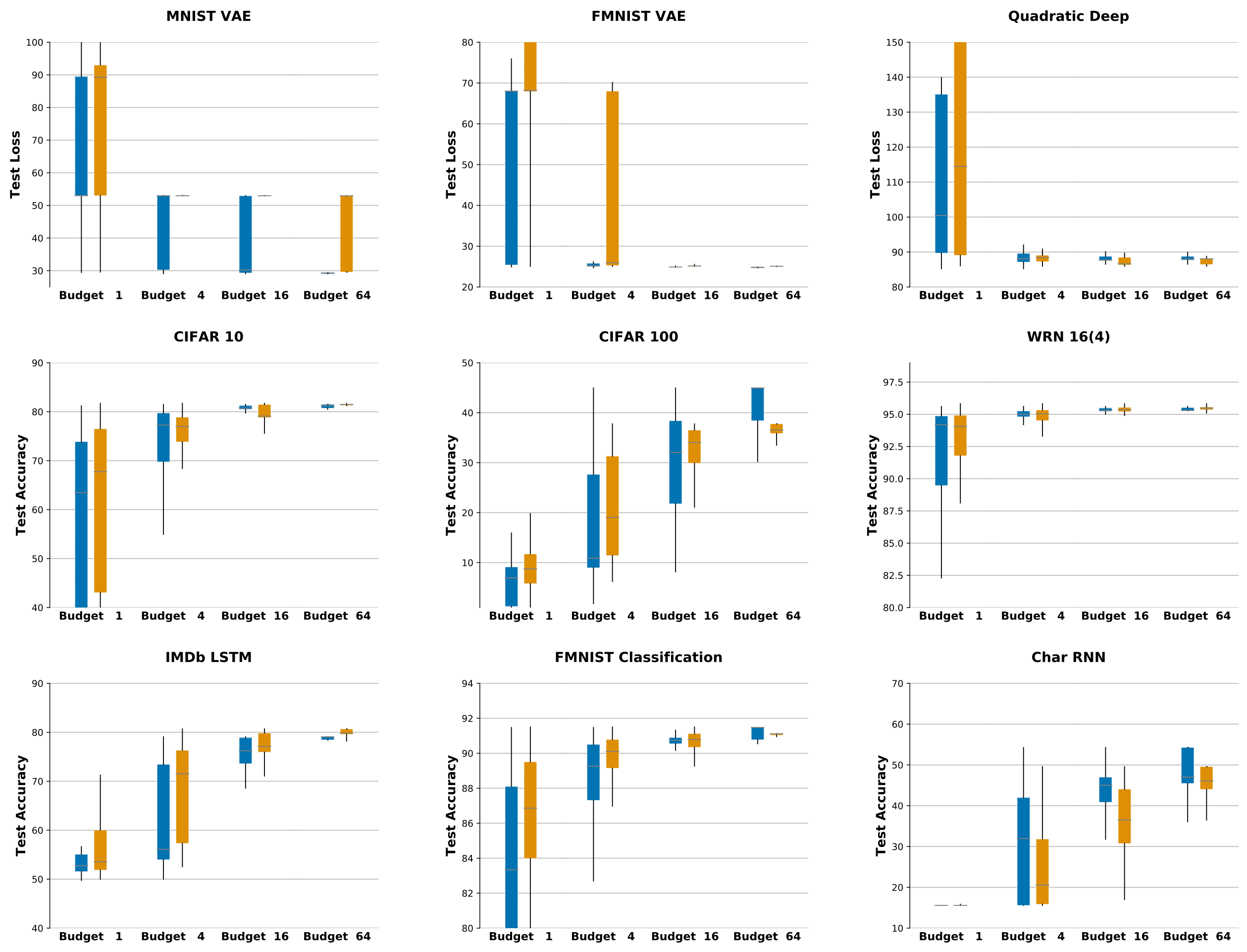


Figure 9: Performance of **SGD-MW**, **SGD-LR$_{\text{eff}}$**, at various hyperparameter search budgets. Image is best viewed in color. Some of the plots have been truncated to increase readability.

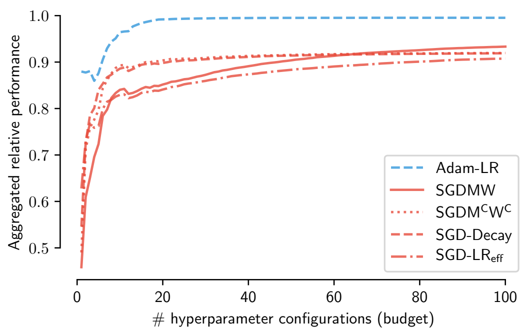


Figure 10: Aggregated relative performance of SGD-LR$_{\text{eff}}$ compared to other optimizers.