Fast Optimizer Benchmark

Simon Blauth[*,1] Tobias Bürger[*,1] Zacharias Häringer[*,1] Jörg K.H. Franke[1] Frank Hutter[2,1]

[*]These authors contributed equally to this work, sharing first authorship
[1]University of Freiburg
[2]ELLIS Institute Tübingen

Abstract In this paper, we present the Fast Optimizer Benchmark (FOB), a tool designed for evaluating deep learning optimizers during their development. The benchmark supports tasks from multiple domains such as computer vision, natural language processing, and graph learning. The focus is on convenient usage, featuring human-readable YAML configurations, SLURM integration, and plotting utilities. FOB can be used together with existing hyperparameter optimization (HPO) tools as it handles training and resuming of runs. The modular design enables integration into custom pipelines, using it simply as a collection of tasks. We showcase an optimizer comparison as a usage example of our tool. FOB can be found on GitHub: https://github.com/automl/FOB.

Keywords: benchmark, deep learning, neural network, training algorithms, optimizer

1 Introduction

Deep learning training algorithms, often simply called *optimizers*, play an integral role in the training process of modern AI models. Therefore, improving optimizers' development is a valuable contribution to the research community. One of the challenges of developing new optimizers is to measure their performance accurately. Researchers need fast, reliable, and reproducible performance measures starting from the early stages of the development cycle.

Our aim with this tool is to provide a first indication whether a new optimizer has potential. To achieve this, we offer a user-friendly development platform that can be easily extended to meet individual needs when comparing different optimizers. We provide tasks from different domains, baseline optimizers, and a convenient way to run experiments with our YAML configuration workflow. Other convenience features like SLURM integration and plotting utilities are also included. Results can be reproduced by sharing the configuration YAML file. A fixed collection of tasks is not only necessary for an optimizer benchmark but is also a valuable resource for other applications. Due to the modular design, the provided tasks can be reused in custom training pipelines or together with existing HPO tools. The tool is also convenient to set up because it does not require dataset registration, cutting down on initial setup time.

1.1 Related Work

In this section, we review several existing benchmarks that are relevant to our work.
AlgoPerf [Dahl et al., 2023] is an extensive optimizer benchmark that focuses on large tasks and implements a time-to-result benchmark with fixed hardware requirements. However, it has limitations due to the necessity for dataset registration and its restriction to large-scale tasks only. For a competitive benchmark result it is still the better option.
MLPerf [Mattson et al., 2020] includes a variety of large tasks but lacks a unified structure to compare optimizers independently of the tasks. Each task requires a separate setup and execution, making it less convenient for benchmarking optimizers.
DeepOBS [Schneider et al., 2019] was an early benchmark, but except for one it only features very

 © 2024 the authors, released under CC BY 4.0

small tasks. Additionally, it is now considered obsolete with AlgoPerf being its successor.
Deep Learning Benchmark Suite[1] only features image datasets and focuses on evaluating accelerators and frameworks rather than optimizers.
Benchopt [Moreau et al., 2022] is a benchmarking suite with a focus on machine learning. It also features one small deep learning task and uses yaml configurations similar to FOB.

1.2 Measuring Optimizer Performance

Optimizer performance can be measured in two primary ways.
The first method focuses on the *time to achieve a certain score*, either a predefined performance threshold or the peak performance of the training run. The second method involves *fixing the training duration* and then measuring the final performance. This can be done by setting a fixed number of epochs or by limiting the wall time.
Another important aspect is the optimizer's sensitivity to its hyperparameters. Optimizers with good defaults that are easy to tune are more desirable, as they reduce the need for extensive tuning and make it easier to achieve optimal performance across different tasks.

The decision for FOB is to train for a fixed number of epochs and then report the best and last performance. This strategy ensures that scores are available even for underperforming optimizers or those with suboptimal hyperparameter configurations. By using epochs instead of wall time, we maintain hardware independence, although this approach can be exploited by optimizers with heavy computations. The chosen number of epochs also influences the outcome: a tight limit favors faster optimizers, while a more generous limit favors those achieving better peak performance.

2 Methods

Our tool is built upon the deep learning framework PyTorch Lightning [Falcon and The PyTorch Lightning team, 2019]. It follows the same abstraction idea and organizes essential deep learning components into exchangeable and extensible modules. The two key modules are *Tasks* and *Optimizers*. Training and evaluation are handled by the core engine. Users can decide which task-optimizer pair to train; they can configure training parameters and optimizer hyperparameters, and specify some model parameters. Configurations are written in human-readable YAML files. Users can quickly assess their results with heatmaps that are being plotted after the evaluation of the model finishes.

2.1 Optimizer

Besides optimizers, learning rate schedulers are important for adjusting parameter updates during training. In this benchmark, we treat both together as the optimizer being evaluated. A small collection of optimizers has been implemented as baselines, and to reproduce results from literature references. In addition to SGD with Momentum [Sutskever et al., 2013] and Adafactor [Shazeer and Stern, 2018], the baselines include implementations of AdamW [Loshchilov and Hutter, 2019] and AdamCPR [Franke et al., 2023], all with a cosine annealing scheduler [Loshchilov and Hutter, 2017].

2.1.1 Adding an Optimizer

In an optimizer benchmark, it is a common use case to add your own optimizer and compare it against a baseline. This process was designed to be easy and convenient. One can add their own implementation of an optimizer by simply following the template in the repository. The Python code is added in a single file and no further understanding of the complete code base is required. The user has to implement a function with the following API.

```
configure_optimizers(model: GroupedModel, config: OptimizerConfig)
    -> OptimizerLRScheduler:
```

[1] https://github.com/HewlettPackard/dlcookbook-dlbs

2.2 Tasks

The benchmarking process involves multiple tasks, each comprising a predefined model architecture, training, validation, and test datasets, along with specified performance metrics. As this benchmark aims to be budget-friendly, each task must be appropriately sized. Training times are constrained to ensure completion of the full suite within one day on a single node with 4 GPUs. The tasks cover a range of domains, model architectures, and sizes to ensure generalization and fair comparisons. Popular domains, such as computer vision and natural language processing, have been allocated the largest share of training time.

The *MNIST, Classification Small, Classification* and *Segmentation* tasks from the **computer vision** domain utilize the MNIST, CIFAR-100, ImageNet-64, and ADE20K datasets [Chrabaszcz et al., 2017; Krizhevsky, 2009; Lecun et al., 1998; Zhou et al., 2017]. We employ models such as MLP, ResNet, Wide-ResNet, and SegFormer [He et al., 2016; Xie et al., 2021; Zagoruyko and Komodakis, 2016].

In the *Translation* task from the **natural language processing** domain, the goal is to translate between English and German sentences from the WMT17 dataset [Macháček and Bojar, 2013] using a small T5 transformer [Raffel et al., 2020].

For the *Graph Tiny* and *Graph* tasks, we utilize the Cora dataset and the molecular ogb-molhiv dataset [Hu et al., 2020; McCallum et al., 2000]. We leverage **graph** neural network models such as a simple Graph Convolutional Network (GCN) and a Graph Isomorphism Network (GIN) [Kipf and Welling, 2017; Xu et al., 2019].

Lastly the *Tabular* task employs a FT-Transformer on the **tabular** California Housing dataset [Gorishniy et al., 2021; Kelley Pace and Barry, 1997].

Dataset	Model	Problem Type	GPUs	Runtime
MNIST	MLP	Image Classification	1	1 min
Cora	GCN	Node Classification	1	1 min
California Housing	FT Transformer	Tabular Regression	1	2 min
CIFAR100	Resnet18	Image Classification	1	10 min
ogbg-molhiv	GIN	Graph Property Prediction	1	20 min
Imagenet-64x64	Wide ResNet	Image Classification	4	4 h
MIT Scene Parse	SegFormer	Semantic Segmentation	4	5 h
WMT17(en-de)	T5 small	Machine Translation	4	6 h

Table 1: The ensemble of tasks in FOB. Runtime was measured on 4xA100 40GB GPUs, 128GB RAM. For comprehensive details, refer to the repository.

2.2.1 Modifying a Task.

Minor modifications can be applied to the models used, with certain parameters already accessible through the YAML configuration files. Common examples include adjustments to train transformations, dropout rates, and hidden channel sizes. Furthermore, users can easily adapt existing tasks by making a single code change and adding an entry to the `default.yaml` file, such as modifying the activation function.

Users can also introduce their own tasks, enabling coverage of more domains where specific optimizers may excel. To facilitate user contributions, the repository includes a template and instructions for adding custom tasks.

2.3 Usage

Our tool tries to cover a wide range of use cases. We want users to be able to get started quickly but also give them the freedom to integrate it into their existing workflows.

2.3.1 Running Experiments.

Users specify their experimental setup through a single YAML configuration file. Once the configuration is set, the tool handles all other aspects of running the experiment, including data loading, model training, and evaluation. This makes it easy to set up and run experiments without needing to write any code. FOB also has a built-in grid-search functionality to explore hyperparameters. Tasks and optimizers come with sensible default values, and users have the option to overwrite them as needed by simply including them in their experiment files. We suggest referring to the corresponding default YAML files to explore all available configurable options. The repository contains several helpful examples of experiments for reference.

2.3.2 Integration with Existing Tools.

For more advanced HPO, users can integrate FOB with existing HPO tools like SMAC [Lindauer et al., 2022] or Optuna [Akiba et al., 2019]. Although advanced HPO is not built into FOB, its engine supports training and checkpointing, allowing users to resume runs. Examples using SMAC are provided in the repository, showcasing how to implement this integration. In section 3, we present experimental results that were obtained using SMAC.

2.3.3 Using Tasks in Your Own Pipeline.

The modular design of FOB allows researchers to use it purely as a collection of tasks and integrate them into their own frameworks or benchmarking setups. This enables researchers to benefit from FOB's task collection without being constrained by the tool's architecture. For pointers on how you might do this see appendix C. An example of this use case is available in our repository, demonstrating how to use FOB tasks with NePS [Stoll et al., 2023].

3 Experiments

We conducted experiments using SMAC [Lindauer et al., 2022] on four of our smaller tasks to evaluate the performance of three optimizers: AdamW, AdamCPR, and SGD [Franke et al., 2023; Loshchilov and Hutter, 2019; Sutskever et al., 2013]. For experiments utilizing our full suite of tasks, refer to appendix D.

The hyperparameters of each optimizer were tuned using SMAC, see appendix A for the search space. Multi-Fidelity-Optimization was used with the training epochs as budget. The intensifier used was Hyperband [Li et al., 2018] with $\eta = 3$. Each optimizer was tuned for 250 trials with 10% initial configurations. For each optimizer, we took the best configuration returned by SMAC and retrained them on three seeds $s \in \{1, 2, 3\}$ to obtain the test performance detailed in table 2. For each task and optimizer, we report the test performance of the last model checkpoint as well as the test performance of the model checkpoint with the best validation performance. The models were trained using A100 and A40 GPUs on the HELIX cluster.

Task	Class. Small	Graph	Graph Tiny	Tabular
Metric	Accuracy ↑	ROC AUC ↑	Accuracy ↑	RMSE ↓
AdamCPR best	77.2 ± 0.16	75.57 ± 3.13	80.27 ± 0.32	0.403 ± 0.011
AdamW best	**77.29 ± 0.17**	75.75 ± 1.04	80.63 ± 1.0	0.401 ± 0.008
SGD best	77.27 ± 0.28	75.89 ± 0.58	14.47 ± 3.45	0.419 ± 0.007
AdamCPR last	77.1 ± 0.2	**76.86 ± 1.06**	79.5 ± 0.61	**0.399 ± 0.005**
AdamW last	77.18 ± 0.14	75.86 ± 0.35	**81.4 ± 0.35**	0.401 ± 0.005
SGD last	77.0 ± 0.49	75.99 ± 1.34	14.83 ± 3.67	0.421 ± 0.008

Table 2: Performance per task of best hyperparameter configuration found by SMAC evaluated on three seeds.

3.1 Discussion

On the *Classification Small* task, all optimizers performed similarly, with AdamW slightly outperforming the others. For the *Graph* task, AdamCPR last achieved the highest performance, demonstrating an advantage over the other optimizers. On the *Graph Tiny* task, AdamW achieved the highest score, while SGD performed poorly. In the *Tabular* task, the results were similar across all optimizers, with AdamCPR achieving the best result.

Overall, both AdamW and AdamCPR performed similarly, obtaining consistent results across the tasks. In contrast, SGD exhibited clear weaknesses, particularly in the *Graph Tiny* and *Tabular* tasks, where its performance was significantly lower than that of AdamW and AdamCPR.

4 Broader Impact & Limitations

Large-scale benchmarks can require high computational efforts. We aim to mitigate this issue by including small tasks in our benchmark suite. When running the entire suite, especially with many hyperparameter configurations, the computations can still be expensive. For example our experiment detailed in appendix D required approximately 2400 GPU hours on A100 GPUs. With an estimated carbon efficiency of 0.432 kg of CO2 per kWh, the total carbon emission from this experiment is approximately 260 kg of CO2. In contrast, our experiment in section 3, which only used smaller tasks, required approximately 165 GPU hours and 18 kg of CO2. This shows that the inclusion of smaller tasks with a modular approach enables researchers to conduct experiments with smaller environmental footprints. We do not expect any new societal risk to arise from this work since we only use already existing models and publicly available datasets.

The limitations of our contribution include the absence of very large tasks in our benchmark, which might lead to some aspects not being captured correctly. However, there are already tools that achieve this (e.g. AlgoPerf [Dahl et al., 2023]). Also, some fields of deep learning like speech recognition are not covered. Since our tool is designed to be extensible, we hope that these gaps might be filled in the future. Since we built our tool upon PyTorch Lightning [Falcon and The PyTorch Lightning team, 2019], we are restricted to the hardware supported by that framework, so some hardware might not work with FOB.

5 Conclusion

In this paper, we introduced the **Fast Optimizer Benchmark (FOB)**, a convenient and extensible tool designed for evaluating deep learning optimizers across diverse tasks. FOB supports tasks from multiple domains, offers easy configuration through YAML files, and includes features such as SLURM integration to support distributed computing on clusters. Additionally, the modular design of FOB facilitates its use for various purposes, including hyperparameter optimization (HPO), by allowing easy integration with other tools. We showcased the usage of FOB and conducted experiments to demonstrate its capabilities.

An important open question remains for future work: To what degree do the results of this benchmark correlate with larger, related benchmarks? Establishing a reliable and cost-effective proxy for extensive benchmarks like AlgoPerf would greatly benefit the research community.

Acknowledgements. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

References

Akiba, T. et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. Ed. by A. Teredesai et al. ACM, pp. 2623–2631. URL: https://doi.org/10.1145/3292500.3330701.

Chrabaszcz, P., I. Loshchilov, and F. Hutter (2017). *A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets*. arXiv: 1707.08819 [cs.CV].

Dahl, G. E. et al. (2023). *Benchmarking Neural Network Training Algorithms*. arXiv: 2306.07179 [cs.LG].

Falcon, W. and The PyTorch Lightning team (2019). *PyTorch Lightning*. Version 1.4. URL: https://github.com/Lightning-AI/lightning.

Franke, J. K. H. et al. (2023). *Constrained Parameter Regularization*. arXiv: 2311.09058 [cs.LG].

Gorishniy, Y. et al. (2021). "Revisiting Deep Learning Models for Tabular Data". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by M. Ranzato et al., pp. 18932–18943. URL: https://proceedings.neurips.cc/paper/2021/hash/9d86d83f925f2149e9edb0ac3b49229c-Abstract.html.

He, K. et al. (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, pp. 770–778. URL: https://doi.org/10.1109/CVPR.2016.90.

Hu, W. et al. (2020). "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html.

Kelley Pace, R. and R. Barry (1997). "Sparse spatial autoregressions". In: *Statistics & Probability Letters* 33.3, pp. 291–297. URL: https://www.sciencedirect.com/science/article/pii/S016771529600140X.

Kipf, T. N. and M. Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SJU4ayYgl.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*. Tech. rep.

Lecun, Y. et al. (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86, pp. 2278–2324.

Li, L. et al. (2018). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.185, pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html.

Lindauer, M. et al. (2022). "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *J. Mach. Learn. Res.* 23, 54:1–54:9. URL: http://jmlr.org/papers/v23/21-0888.html.

Loshchilov, I. and F. Hutter (2017). "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Skq89Scxx.

Loshchilov, I. and F. Hutter (2019). "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Macháček, M. and O. Bojar (2013). "Results of the WMT13 Metrics Shared Task". In: *Proceedings of the Eighth Workshop on Statistical Machine Translation*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 45–51. URL: https://aclanthology.org/W13-2202.

Mattson, P. et al. (2020). "MLPerf Training Benchmark". In: *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by I. S. Dhillon, D. S. Papailiopoulos, and V. Sze. mlsys.org. URL: https://proceedings.mlsys.org/book/309.pdf.

McCallum, A. K. et al. (2000). "Automating the Construction of Internet Portals with Machine Learning". In: *Information Retrieval* 3.2, pp. 127–163. URL: https://doi.org/10.1023/A:1009953814988.

Moreau, T. et al. (2022). "Benchopt: Reproducible, efficient and collaborative optimization benchmarks". In: *NeurIPS*. URL: https://arxiv.org/abs/2206.13424.

Raffel, C. et al. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21, 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html.

Schneider, F., L. Balles, and P. Hennig (2019). "DeepOBS: A Deep Learning Optimizer Benchmark Suite". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=rJg6ssC5Y7.

Shazeer, N. and M. Stern (2018). "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4603–4611. URL: http://proceedings.mlr.press/v80/shazeer18a.html.

Stoll, D. et al. (2023). *Neural Pipeline Search (NePS)*. Version 0.11.0. URL: https://github.com/automl/neps.

Sutskever, I. et al. (2013). "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1139–1147. URL: http://proceedings.mlr.press/v28/sutskever13.html.

Xie, E. et al. (2021). "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by M. Ranzato et al., pp. 12077–12090. URL: https://proceedings.neurips.cc/paper/2021/hash/64f1f27bf1b4ec22924fd0acb550c235-Abstract.html.

Xu, K. et al. (2019). "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=ryGs6iA5Km.

Zagoruyko, S. and N. Komodakis (2016). "Wide Residual Networks". In: *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. Ed. by R. C. Wilson, E. R. Hancock, and W. A. P. Smith. BMVA Press. URL: http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

Zhou, B. et al. (2017). "Scene Parsing through ADE20K Dataset". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 5122–5130. URL: https://doi.org/10.1109/CVPR.2017.544.

Submission Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes] See section 4

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See section 4

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? https://2022.automl.cc/ethics-accessibility/ [Yes]

2. If you ran experiments…

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] See section 3.

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] All of the experimental details are either specified in the paper or in our repository.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] Some experiments were conducted on multiple seeds see appendix D. Others had only one seed, see section 3. We always disclose the seeds used.

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] Where applicable, see appendix D.

   (e) Did you report the statistical significance of your results? [No] We did not report the statistical significance explicitly. However, we mention whether results are statistically significant in our discussion, see appendix D.3.

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A] Our contribution contains a tabular dataset as well as other real datasets. The evaluation was achievable using real datasets, eliminating the need for surrogate ones.

   (g) Did you compare performance over time and describe how you selected the maximum duration? [Yes] We monitored the training with tensorboard and selected training durations manually. We tried to keep training as short as possible while still achieving reasonable performance.

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See table 1 and section 4

   (i) Did you run ablation studies to assess the impact of different components of your approach? [N/A] Our tool offers a huge range of customization options. Running ablation studies on all of them is not feasible.

3. With respect to the code used to obtain your results…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] See repository.

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] See repository.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] See repository.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] See release in our repository, we added the raw results as a zip file.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] See repository.

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes]

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] We used publicly available datasets.

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We use only publicly available datasets and existing models.

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] See repository.

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] See repository.

6. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not use crowdsourcing or conducted research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not use crowdsourcing or conducted research with human subjects.

7. If you included theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? [N/A] We did not include theoretical results.

(b) Did you include complete proofs of all theoretical results? [N/A] We did not include theoretical results.

A Hyperparameter Search Spaces

The searchspace detailed in table 3 was used as an input for SMAC [Lindauer et al., 2022]. For more details on the experiment refer to section 3.

Hyperparameter	AdamW	AdamCPR	SGD
Learning Rate	Log [1e-5, 1e-1]	Log [1e-5, 1e-1]	Log [1e-5, 1e-1]
Minimal LR	Log [0.1%, 10%]	Log [0.1%, 10%]	Log [0.1%, 10%]
LR Warmup	Log [0.1%, 100%]	Log [0.1%, 100%]	Log [0.1%, 100%]
Weight Decay	Log [1e-5, 1]	-	Log [1e-5, 1]
$1 - \beta_1$	Log [1e-2, 2e-1]	Log [1e-2, 2e-1]	-
β_2	[0.9, 0.999]	[0.9, 0.999]	-
Momentum	-	-	[0, 1]
Kappa Init Param	-	Log [1, 19550]	-
Kappa Init Method	-	warm_start	-

Table 3: The respective hyperparameter search spaces for the AdamW, AdamCPR and SGD optimizers used in the SMAC experiment.

B Usage Example: YAML configuration

Here we provide some examples of how a user can configure FOB using YAML configuration files.

```yaml
task:
  name: mnist
  max_epochs: 10
  model:
    num_hidden: 42
optimizer:
  name: adamw_baseline
  learning_rate: 1.0e-2
```

Training on the *MNIST* task for 10 epochs. It is possible to make some changes to the model like the number of hidden units. AdamW is used as an optimizer and the learning rate is set. All parameters that are not specified in the configuration are taken from a default file.

Figure 1: A YAML configuration to start a small MNIST experiment.

```yaml
task:
  name: mnist
  max_epochs: 10
  model:
    num_hidden: [128, 256]
optimizer:
  - name: adamw_baseline
    beta2: 0.98
  - name: sgd_baseline
    momentum: 0.5
engine:
  seed: [42, 47]
```

This example showcases how we can specify lists in the YAML file to perform a gridsearch. FOB essentially computes the cartesian product of all lists in an experiment configuration. In this example we have 2 model sizes, 2 optimizers and 2 seeds, resulting in 8 training runs in total.

Figure 2: A YAML configuration to compare different settings on the MNIST task.

```yaml
task:
  name: classification_small
optimizer:
  - name: adamcpr_fast
    learning_rate:
      [1.e-1, 1.e-2, 1.e-3, 1.e-4]
    kappa_init_param:
      [1, 2, 4, 8, 16]
  - name: adamw_baseline
    learning_rate:
      [1.e-1, 1.e-2, 1.e-3, 1.e-4]
    weight_decay:
      [10, 1, 1.e-1, 1.e-2, 1.e-3]
engine:
  seed: [1, 2, 3]
evaluation:
  output_types: [pdf, png]
  plot:
    x_axis:
      - optimizer.kappa_init_param
      - optimizer.weight_decay
```

This is the configuration used to perform the experiment of fig. 6. Here we have 2 optimizers each evaluated for 4x5 hyperparameter configurations on 3 seeds each for a total of 120 trials. We also showcase our plotting utilities which are specified in the same YAML file. Running this file directly results in fig. 6.

Figure 3: The YAML configuration used to create the comparison on *Classification Small* in fig. 6.

C Python example

Below we give you a starting point if you want to handle training yourself but still want to make use of the tasks from FOB. This snippet shows that given a configuration (like the result from parsing a YAML file) you can easily access the Lightning Module and Datamodule comprising the task.

```python
from pytorch_fob import Engine
# create your config, e.g.
config = {"task": "mnist"}
engine = Engine()
engine.parse_experiment(config)
run = next(engine.runs())
model, datamodule = run.get_task()
# run your custom training pipeline...
```

Figure 4: Using FOB tasks in your own pipeline.

D Case Study: comparing AdamCPR and AdamW

To demonstrate the benefits of our contribution, we include a case study of comparing the **AdamW** [Loshchilov and Hutter, 2019] and the **AdamCPR** [Franke et al., 2023] optimizer on our set of tasks.

D.1 Experimental Setup

We use a Cosine-Annealing learning rate scheduler with linear warmup. The minimum learning rate of the cosine decay is set to 1% of the initial learning rate. For all tasks, we perform a gridsearch over two hyperparameters. Both optimizers sweep over the learning rate, while the second hyperparameter depends on the optimizer: weight decay for AdamW and the Kappa-Init-Parameter for AdamCPR. We choose the 'warm-start' (Kappa-I_s) setting for AdamCPR, so the Kappa-Init-Parameter corresponds to the number of steps until fixing the regularization. We

Task	Classification	Class. Small	MNIST	Segmentation
Metric	Accuracy ↑	Accuracy ↑	Accuracy ↑	mIoU ↑
AdamCPR best	69.15 ± 0.07	77.54 ± 0.16	97.29 ± 0.19	**35.59 ± 0.21**
AdamW best	**69.79 ± 0.04**	**77.87 ± 0.25**	97.61 ± 0.08	35.52 ± 0.12
AdamCPR last	69.22 ± 0.08	77.54 ± 0.16	97.34 ± 0.19	35.35 ± 0.24
AdamW last	69.67 ± 0.1	77.79 ± 0.25	**97.63 ± 0.08**	35.45 ± 0.19

Table 4: Maximal performance of the computer vision tasks gridsearch. All values are given in %.

Task	Graph	Graph Tiny	Tabular	Translation
Metric	ROC AUC ↑	Accuracy ↑	RMSE ↓	BLEU ↑
AdamCPR best	76.72 ± 2.12	81.93 ± 0.72	**0.396 ± 0.004**	**26.4 ± 0.1**
AdamW best	77.18 ± 1.4	**82.03 ± 0.64**	0.398 ± 0.006	26.25 ± 0.22
AdamCPR last	76.42 ± 0.45	79.9 ± 0.7	0.398 ± 0.005	26.4 ± 0.1
AdamW last	**77.4 ± 1.31**	80.1 ± 0.82	0.397 ± 0.005	26.25 ± 0.22

Table 5: Maximal performance of the gridsearch on the *Graph*, *Graph Tiny*, *Tabular* and *Translation* tasks. All values except for RMSE are given in %.

implemented the Kappa-Init-Parameter as a factor of the steps for the learning rate warmup. For the number of learning rate warmup steps, we use 1% of the total steps. This means that the Kappa-Init-Parameter can be a value between 0 and 100.
For the choice of values in the search grid we follow [Franke et al., 2023, Section 5.2] and use values evenly spaced on a $\log_{10}$ scale for learning rate and weight decay, while the values for the Kappa-Init-Parameter are evenly spaced on a $\log_2$ scale. We usually started with a small grid and expanded it in the directions where the best values were at the borders.
The models were trained using A100 and A40 GPUs on the HELIX cluster. For the exact details please refer to the baseline configurations in the repository.

D.2 Results

Here we present the results of our experiments grouped by each task. All experiments were repeated with three seeds $s \in \{1, 2, 3\}$, over which we computed the standard deviation. For each task and optimizer, we report the test performance of the last model checkpoint as well as the test performance of the model checkpoint with the best validation performance.
For each task, we present plots of the performance across the entire search grid. The plots show the test performance of the checkpoint with the highest validation performance, essentially an early-stopping setting.

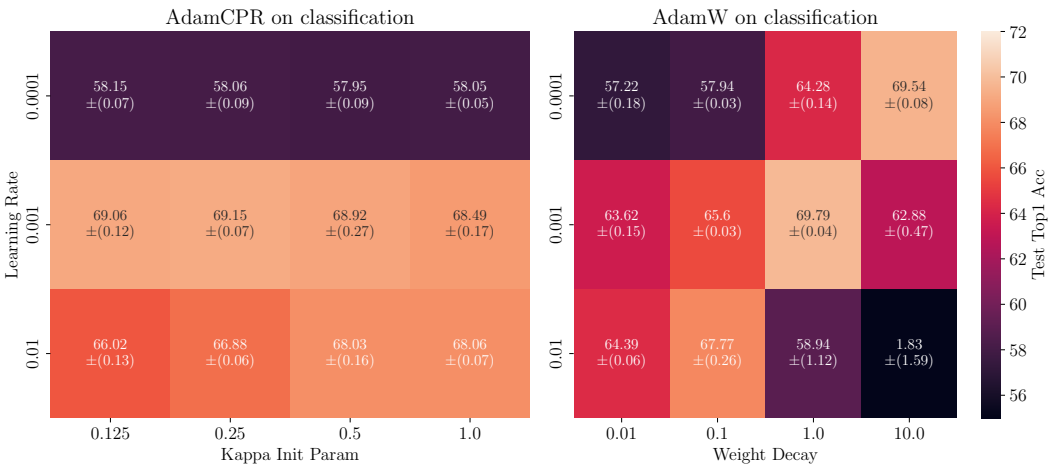

Figure 5: AdamW needs high weight decay regularization.
AdamCPR performs best with few warmup steps.

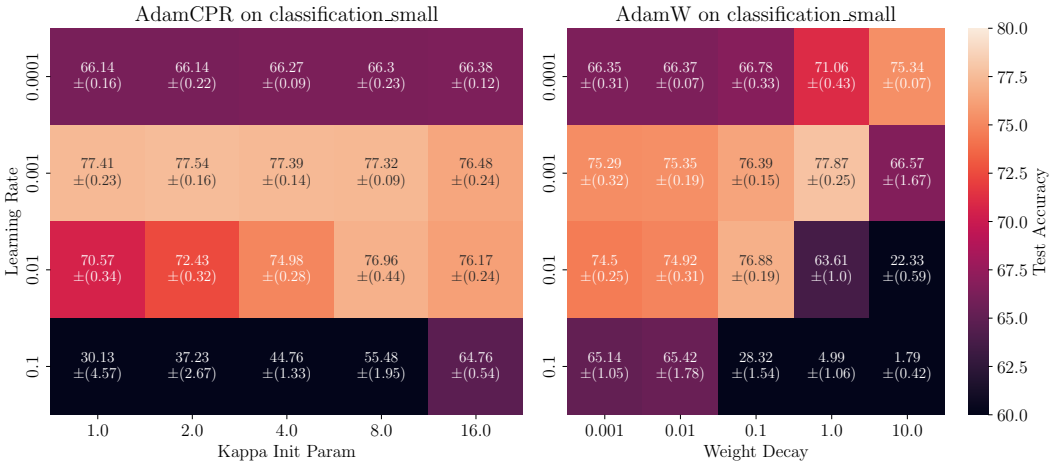

Figure 6: The best hyperparameters for AdamW are the same as for the *Classification* task (fig. 5). Generally, the grid elements that overlap between these tasks look similar in terms of relative performance. For AdamCPR we sadly cannot make this comparison as we chose different values between the tasks. However, we can observe that while peak performance is better for AdamW, more configurations reach a high score for AdamCPR.
Note that our hyperparameters are different from the ones used in [Franke et al., 2023]. For a comparison more closely following the original, check appendix D.4.

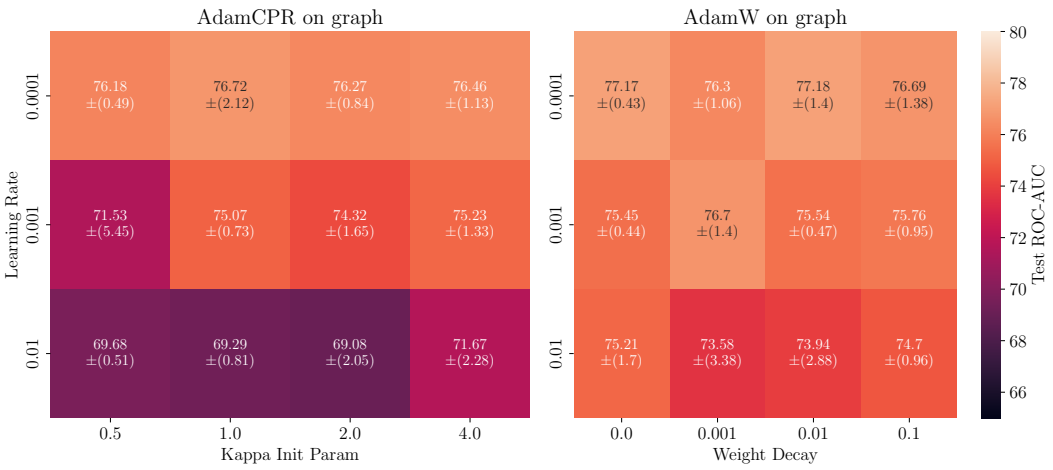

Figure 7: AdamW outperforms AdamCPR on the *Graph* task.

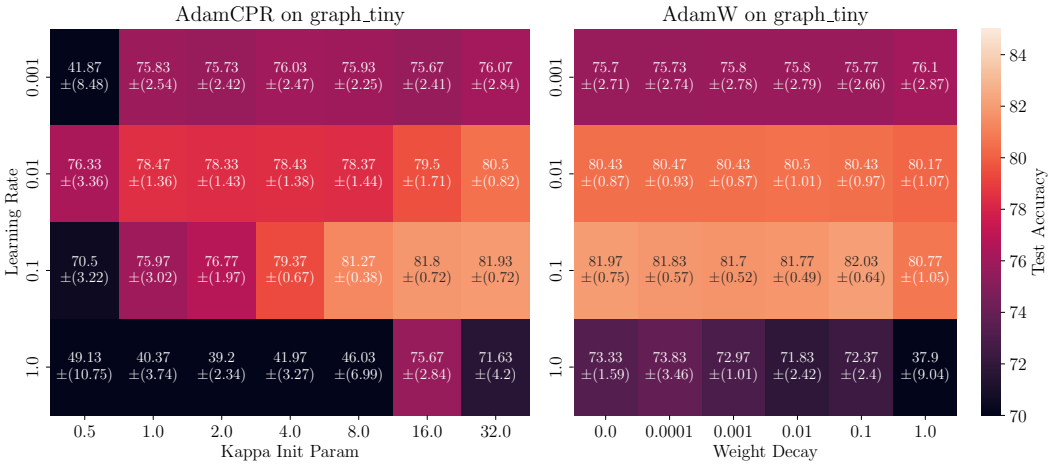

Figure 8: On *Graph Tiny* there is little to no response to weight decay. For AdamW there is almost no change in performance for different weight decay values. AdamCPR performs poorly for this task and converges to AdamW without weight decay for higher Kappa-Init-Parameter (CPR warmup steps).

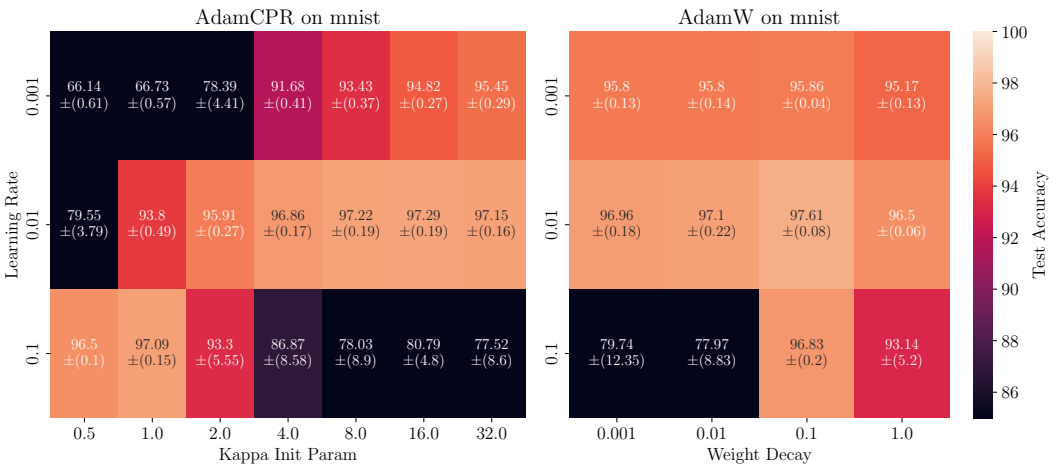

Figure 9: The performance on *MNIST* behaves similar to the one on *Graph Tiny* (8); AdamCPR seems to offer little benefit over plain AdamW without weight decay.

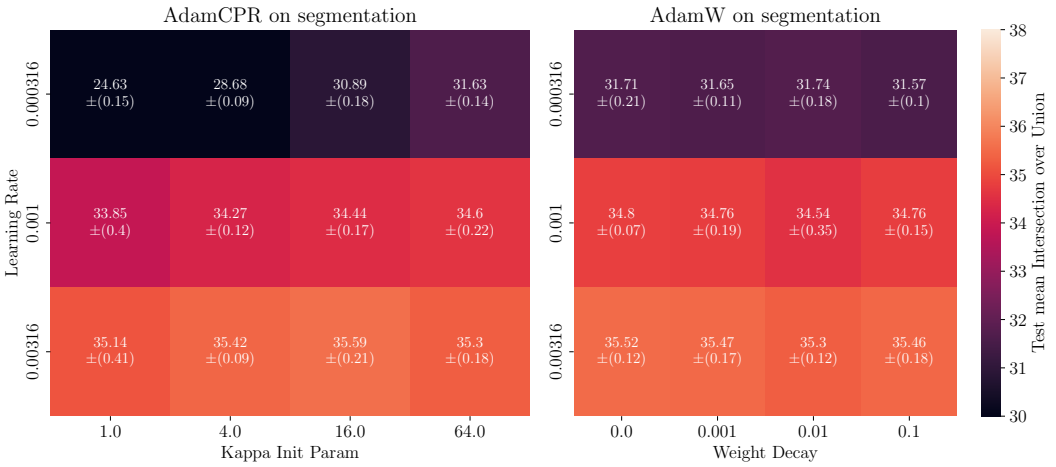

Figure 10: *Segmentation* shows minimal sensitivity to weight decay adjustments and changes in the number of CPR warmup steps. However, AdamCPR performs slightly better here.

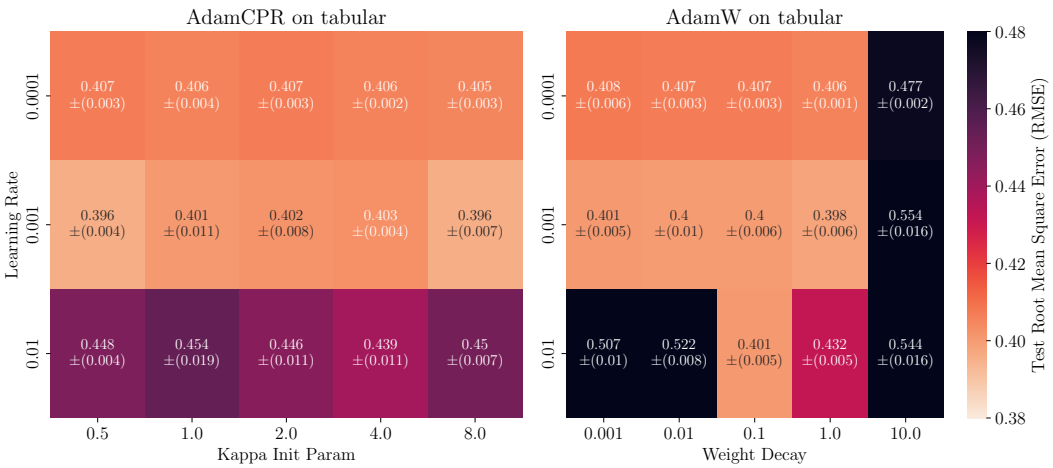

Figure 11: Both optimizers perform well on the `tabular` task, nevertheless AdamCPR performs slightly better.

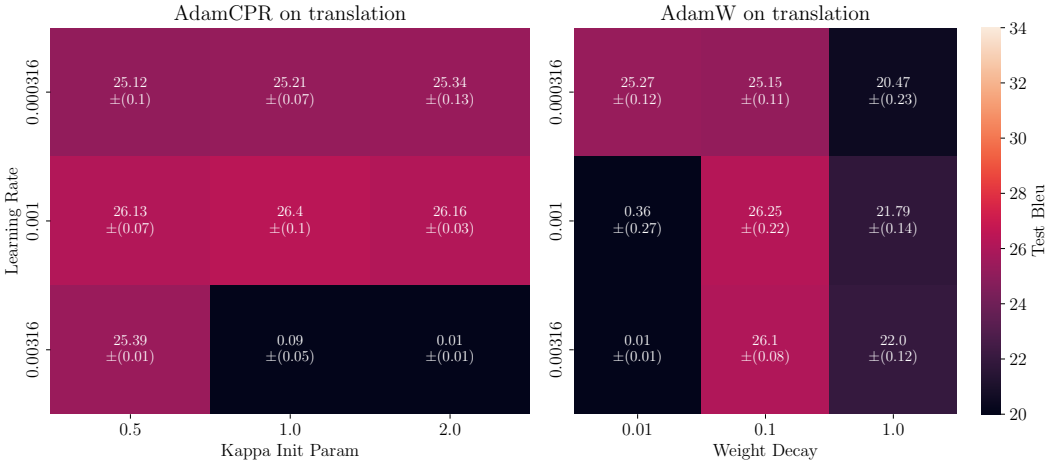

Figure 12: On the *Translation* task AdamCPR outperforms AdamW. However, AdamW shows high responsiveness to weight decay. A better configuration might be found when searching closer to the current optimal weight decay.

D.3 Discussion

When comparing the results of these experiments, one should consider their purpose. They were not solely conducted to compare the two optimizers but were also driven by the goal of exploring the best achievable performance, to set appropriate standards.

In our experiments, the quality of optimizers is primarily evaluated by their peak performance. However, only a limited number of configurations are explored and it is unlikely that a global optimum was found. We assume that the influence of convergence speed is not substantial since the highest validation performance was usually before the final epoch. Reasonable initial search spaces were iteratively expanded to find the best configurations. Thus the sensitivity of the optimizer to its hyperparameters had some influence on the measured peak performance. In the sense that a robust optimizer has a higher chance of finding a good hyperparameter configuration. Since both optimizers heavily depend on their hyperparameters, it was not feasible to use the same

values across all tasks without significantly increasing the experiment budget or sacrificing peak performance.

In our experiments, both optimizers perform similarly. It depends on the task which optimizer has the better score. However, the difference is usually really small, to a point where it probably is negligible and shows little statistical significance. While either shows its own characteristics, we can not determine a superior optimizer based on these results.

D.4 AdamCPR on Classification Small

Here we present a setup close to the experiment in [Franke et al., 2023, Section 5.2]. The hyperparamters were set to match those of the authors and we use a subset of their search grid. Most notably, we set the minimum learning rate to 10% its initial value instead of 1% its initial value as in our experiment (appendix D.1). Also we do not use trivial augment and add `label_smoothing=0.1`. The number of learning rate warmup steps is set to 500.

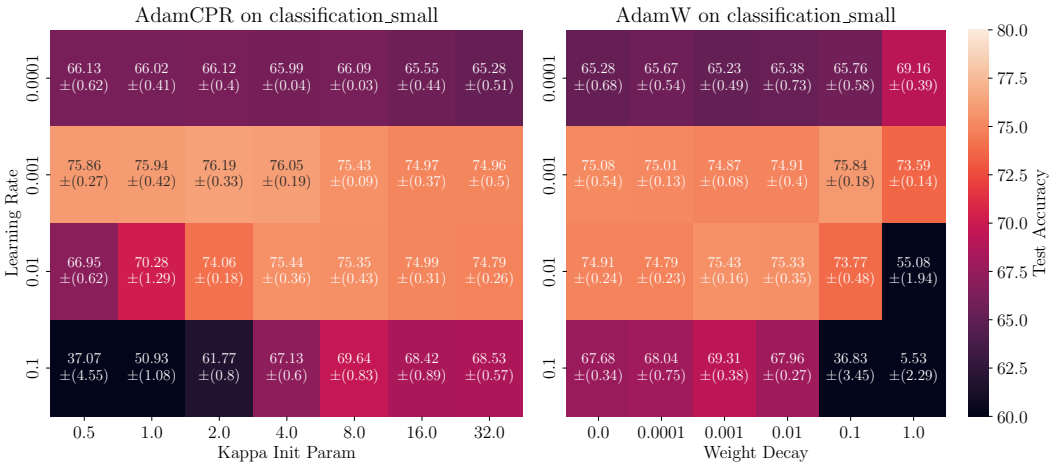

Figure 13: Hyperparameters matching those used in [Franke et al., 2023], resulting accuracies closely match those reported in that work.