Project 3: DNN Hyperparameter Optimization via NNI

Oliver Fowler and Brian Park

North Carolina State University, Computer Science 591/791-025

October 2022

1 HPO Methods

We use NNI to do hyperparameter optimization [1]. We tried multiple tuners such as TPE, Evolution, and Hyperband [2, 3, 4]. We use a simple shallow multi-layer perceptron on MNIST Fashion dataset. This was an example provided by PyTorch. We chose this due to time constraints as we had issues with setting up and allocating nodes on ARC cluster. Other classes were also doing compute intensive assignments, making it difficult to train a more complex model. When we had time left over after doing the required portions, we used a more resource demanding DNN, VGG-19 on CIFAR-10 [5].

2 HPO and Hardware Configurations

2.1 HPO Configurations

For the tuner configurations we used TPE, Evolution, and Hyperband [2, 3, 4]. For hyperparamters to tune, we set the `optimize` mode to `maximize` and let the tuner handle the rest. We set parameters for the learning rate and momentum. For the secondary configuration, we also experimented and treated the number of features and batch size as hyperparameters. Varying both the feature size and batch size will give us a sense of speed. Smaller batches will generally lead to slower training times but higher accuracy, while larger batches will be able to exploit parallelism to get faster training times, but lower accuracy. We capped training at 10 epochs, to conserve experimental training times.

For VGG-19, we repeated the configurations, but number of features is not implemented since we loaded a premade model from PyTorch.

For all configurations, we set the `optimize_mode` to maximize, and the parameter to optimize on would be accuracy.

The configurations for hyperparameters is as follows:

- Learning Rate: Log uniformly sampled between $0.0001, 0.1$
- Momentum: Uniformly sampled between $0, 1$
- Features: $128, 256, 512, 1024$
- Batch Size: $1, 4, 32, 64, 128, 256$

We initially forgot to include different *hyperparameter tuner configurations*. We reran the different hyperparamter configurations for MLP again. We initially ran TPE, Evolution, and Hyperband tuners with their default and minimal configurations. To compare against a different hyperparameter configuration, we reran the same suite of hyperparameter tuners on learning rate, momentum, features, and batch size, but on advanced tuner configurations. The advanced configurations are as follows:

1. TPE
    - Constant Liar Type: Mean
    - Startup Jobs: 10
    - n_{ei} Candidates: 20
    - Linear Forgetting: 50

- Prior Weight: 0.9
- γ: 0.1

2. Evolution
    - Population Size: 50
3. Hyperband
    - R: 100
    - ETA: 25
    - Execute Mode: Parallelism

For reference, the default parameters provided by the API are as follows:

1. TPE
    - Constant Liar Type: Best
    - Startup Jobs: 10
    - n_{ei} Candidates: 24
    - Linear Forgetting: 25
    - Prior Weight: 1.0
    - γ: 0.25
2. Evolution
    - Population Size: 32
3. Hyperband
    - R: 60
    - ETA: 3
    - Execute Mode: Parallelism

2.2 Hardware Configurations

We had difficulties running the hyperparameter tuner experiments successfully on the ARC cluster [6]. We were fortunate enough to get access to the PSC's bridges-2 supercomputer, which uses 8 NVIDIA V100 SXM2 per node, part of a DGX system [7, 8]. The DGX system uses has two Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz CPUs. When allocated via the name `GPU-Shared`, we will only connect to 1 CPU and 1 GPU, as only 4 GPUs are connected via PCIe per CPU node. We use a single GPU, as we're not aware if NNI can utilize multiple GPU in a intra-node setting. Oliver used his RTX 3080 Ti and Brian was able to use his M1 Mac to do the hyperparameter search for some tuners and configuration concurrently. Once we reached an agreement that our code was working and correct for a few configurations, we executed all the configurations on the bridges-2 system for consistency on the V100 GPU in a single batch. The experimental results shown will only be from the V100 GPUs.

3 Experimental Results

3.1 MLP HPO Experiments with Learning Rate and Momentum and Defualt HPO Configurations

Figure 1 shows the result of the MLP with TPE. We observe that Trial 9 has the most optimal parameters with a validation accuracy of 88.26%. The optimal parameters for this trial are a learning rate of 0.0763 and a momentum of 0.5338. We also observe that the duration for all trials is 5:00 minutes. Total time was 5:12 minutes.

Figure 2 shows the result of the MLP with Evolution. We observe that Trial 13 has the most optimal parameters with a validation accuracy of 87.98%. The optimal parameters for this trial are a learning rate of 0.0269 and a momentum of 0.5765. We also observe that the duration for all trials is about 4:47 minutes. Total time was 4:57 minutes.

Table 1: Hyperparameter Summary for MLP on Fashion MNIST

HPO Tuner	Trial	Accuracy	Learning Rate	Momentum	Batch Size	Feature Size
TPE	9	88.26%	0.0763	0.5338	–	–
Evolution	13	87.98%	0.0269	0.5765	–	–
Hyperband	2	88.33%	0.0855	0.6975	–	–
TPE batch	6	88.26%	0.0476	0.4535	64	512
Evolution batch	2	87.98%	0.0794	0.4306	32	512
Hyperband batch	6	88.14%	0.0244	0.7010	32	512
TPE batch (advanced)	10	87.90%	0.0570	0.3596	32	512
Evolution batch (advanced)	10	87.94%	0.0012	0.7361	4	512
Hyperband batch (advanced)	8	87.95%	0.0034	0.7162	4	128

Figure 3 shows the result of the MLP with Hyperband. We observe that Trial 2 has the most optimal parameters with a validation accuracy of 88.33%. The optimal parameters for this trial are a learning rate of 0.0855 and a momentum of 0.6975. We also observe that the duration for all trials is also about 4:44 minutes. Total time was 5:02 minutes.

3.2 MLP HPO Experiments with Learning Rate, Momentum, Feature Size, and Batch Size and Defualt HPO Configurations

Figure 4 shows the result of the MLP with TPE. We observe that Trial 6 has the most optimal parameters with a validation accuracy of 88.26%. The optimal parameters for this trial are a learning rate of 0.0476 and a momentum of 0.4535. The optimal batch size is 32 and the number of features is 512. Duration for each trial ranged from 4-24 minutes. Trial 6 had a execution time of 5:43 minutes. Total time was 24:46 minutes.

Figure 5 shows the result of the MLP with Evolution. We observe that Trial 2 has the most optimal parameters with a validation accuracy of 87.98%. The optimal parameters for this trial are a learning rate of 0.0794 and a momentum of 0.4306. The optimal batch size is 32 and the number of features is 512. Duration for each trial ranged from 4-21 minutes. Trial 2 had a execution time of 5:01 minutes. Total time was 33:20 minutes.

Figure 6 shows the result of the MLP with Hyperband. We observe that Trial 6 has the most optimal parameters with a validation accuracy of 88.14%. The optimal parameters for this trial are a learning rate of 0.0244 and a momentum of 0.7010. The optimal batch size is 64 and the number of features is 512. Duration for each trial ranged from 3-33 minutes. Trial 6 had a execution time of 5:01 minutes. Total time was 5:12 minutes.

3.3 MLP HPO Experiments with Learning Rate, Momentum, Feature Size, and Batch Size and Advanced HPO Configurations

Figure 7 shows the result of the MLP with TPE and advanced configurations. We observe that Trial 10 has the most optimal parameters with a validation accuracy of 87.9%. The optimal parameters for this trial are a learning rate of 0.0570 and a momentum of 0.3596. The optimal batch size is 32 and the number of features is 512. Duration for each trial ranged from 4-24 minutes. Trial 10 had a execution time of 5:37 minutes. Total time was 23:34 minutes.

Figure 8 shows the result of the MLP with Evolution and advanced configurations. We observe that Trial 10 has the most optimal parameters with a validation accuracy of 87.94%. The optimal parameters for this trial are a learning rate of 0.0012 and a momentum of 0.7361. The optimal batch size is 4 and the number of features is 512. Duration for each trial ranged from 4-19 minutes. Trial 10 had a execution time of 9:48 minutes. Total time was 19:43 minutes.

Figure 9 shows the result of the MLP with Hyperband and advanced configurations. We observe that Trial 8 has the most optimal parameters with a validation accuracy of 87.95%. The optimal parameters for this trial are a learning rate of 0.0034 and a momentum of 0.7162. The optimal batch size is 4 and the number of features is 128. Duration for each trial ranged from 4-23 minutes. Trial 8 had a execution time of 10:46 minutes. Total time was 23:30 minutes.

3.4 MLP HPO Experiments Conclusions

For all of the tuners, we are able to see speed differences now once batch size and feature size were added to the search space. Surprisingly Hyperband at the default configurations an search on only learning rate and momentum still performs the best with a peak validation accuracy of 88.33%.

Tree-structured Parzen Estimator (TPE) has the most parameters to tune in its hyperparameter tuner. It's considered the default tuner of NNI, but it has the drawback that it can discover relationships between different hyperparameters unlike others. TPE is an SMBO algorithm. It models $P(x|y)$ and $P(y)$ where x represents hyperparameters and y the evaluation result. $P(x|y)$ is modeled by transforming the generative process of hyperparameters, replacing the distributions of the configuration prior with non-parametric densities [2]. Since we capped the trials at 20, we think we couldn't see the effects for larger search spaces such as adding batch size and feature size. These add on to the search space as well as vary the speed of the training process, since smaller batch sizes are generally slower and can't be parallelized compared to larger batch sizes. The same applies for feature sizes.

For Evolution, NNI API documentation claims that the larger population size, the better evolution performance. It seems like the difference from population size of 32 to 50 doesn't make a difference. Interestingly, it decreased the execution time from 33:20 to 19:42 minutes. It's also reccomended from NNI to make the concurrency lower than the population size, since evolutionary algorithm requires prior knowledge. With a concurrency set to 20 and population of 50, there's more search that Evolution can benefit or iterate on. From the authors, it seems that Evolution search was also used to search for better architecturs [3]. We did not observe this, but it seems like with higher population size, we're able to find a new batch size 4 compared to 32 that can perform about the same in accuracy with Evolution tuner set to default parameter of 32.

The basic idea of Hyperband is to create several buckets, each having n randomly generated hyperparameter configurations, each configuration using r resources. After the n configurations are finished, it chooses the top $\frac{n}{eta}$ configurations and runs them using increased $r \cdot eta$ resources [4]. Maybe more trials and training time is needed to search for better parameters. It seems like the parameters we provided to Hyperband search were not able perform as well as the default, maybe due to less successive halving or not enough trials being fed. Hyperband had the longest individual trial time compared amonst other tuners.

3.5 VGG-19 HPO Experiments with Learning Rate and Momentum

Just in case MLP didn't show interesting results, we also ran a few hyperparameter tuners for VGG-19. We couldn't run the whole suite of configurations as it took very long, so we capped our trials at 5 hours. Figure 10 shows a overall view of our results. Hyperband HPO tuner could not finish, so their results could've been better if ran to completion for a fair evaluation. Evolution HPO tuner had the best hyperparameters with a validation accuracy of 76.15%. We actually crashed our initial run, because it hit OOM. Lowering down the concurrency from 20 to 5 resolved this issue, as VGG-19 is a much larger model and the NVIDIA V100 has only 32GB of HBM memory. The results are still relatively the same across trials. We didn't experiment with batch size though. We wonder if more interesting results would be shown for more search space, longer runtimes, and deeper networks.

4 Lessons Learned

NNI fortunately parallelizes the hyperparameter tuning process by enabling concurrent models to be trained at the same time. Unfortunately, we weren't aware of this when switching between hardware when running experiments. Thus we often crashed our own devices or had our devices struggle to operate due to a configuration suited towards high end GPU like the V100.

We also learned that there are actually many competing hyperparameter optimizers and tuners out there that we were aware of. PyTorch presents Ray Tune in their documentation as a recommendation for hyperparameter tuning [9]. There is also an issue and discussion about what the differences between the two are, which is positioning. We think the main difference is that NNI is a platform that also eases for other methods like pruning and quantization. So much more hyperparameter tuning frameworks exist like HyperOpt, sklearn's GridSearchCV, Optune, Weights & Biases, and many more [10, 11, 12]. Some support classical machine learning algorithms, and other can support deep neural networks. But in the end, the tools seem to all serve a similar purpose with different positioning.

5 GitHub Repository

The GitHub repository for this report is publicly accessible here [13]. To reproduce our findings, please read the `README.md` under the `proj3` directory. If there are any setup issues on bridges-2 supercomputer or locally, please contact bcpark@ncsu.edu.

References

[1] Microsoft, "Neural Network Intelligence," 1 2021.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[3] E. Real, S. Moore, A. Selle, S. Saxena, Y. I. Leon-Suematsu, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *CoRR*, vol. abs/1703.01041, 2017.

[4] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Efficient hyperparameter optimization and infinitely many armed bandits," *CoRR*, vol. abs/1603.06560, 2016.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[6] "Arc: A root cluster for research into scalable computer systems." `https://arcb.csc.ncsu.edu/~mueller/cluster/arc/`.

[7] "Nvidia v100 datasheet." `https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf`.

[8] S. T. Brown, P. Buitrago, E. Hanna, S. Sanielevici, R. Scibek, and N. A. Nystrom, "Bridges-2: A platform for rapidly-evolving and data intensive research," in *Practice and Experience in Advanced Research Computing*, PEARC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[9] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *CoRR*, vol. abs/1807.05118, 2018.

[10] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 115–123, PMLR, 17–19 Jun 2013.

[11] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[13] "Csc 791-025 github repository." `https://github.com/briancpark/csc791-025/tree/main/proj3`.

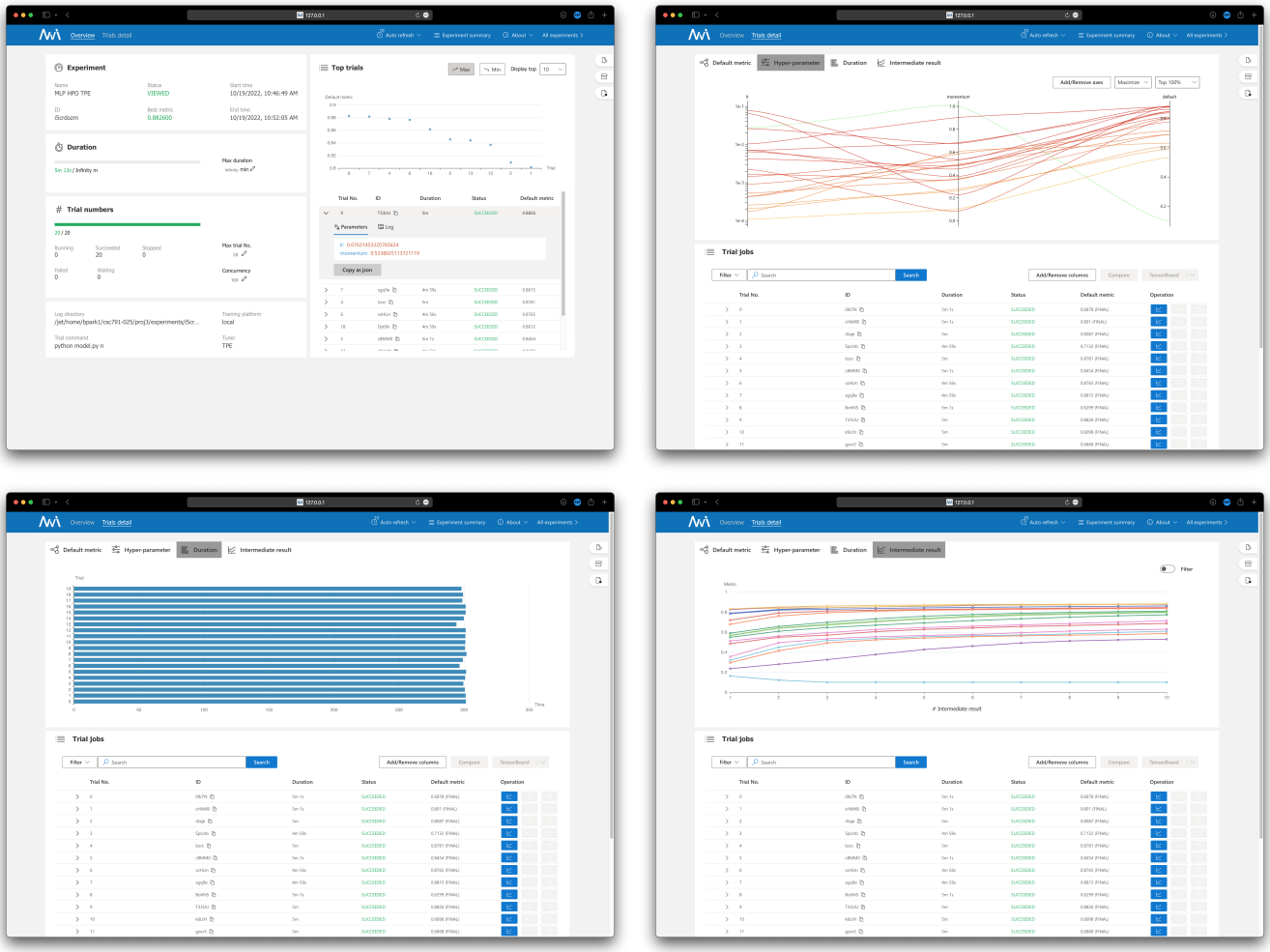

Figure 1: MLP with TPE Tuner on Learning Rate and Momentum

6 Appendix

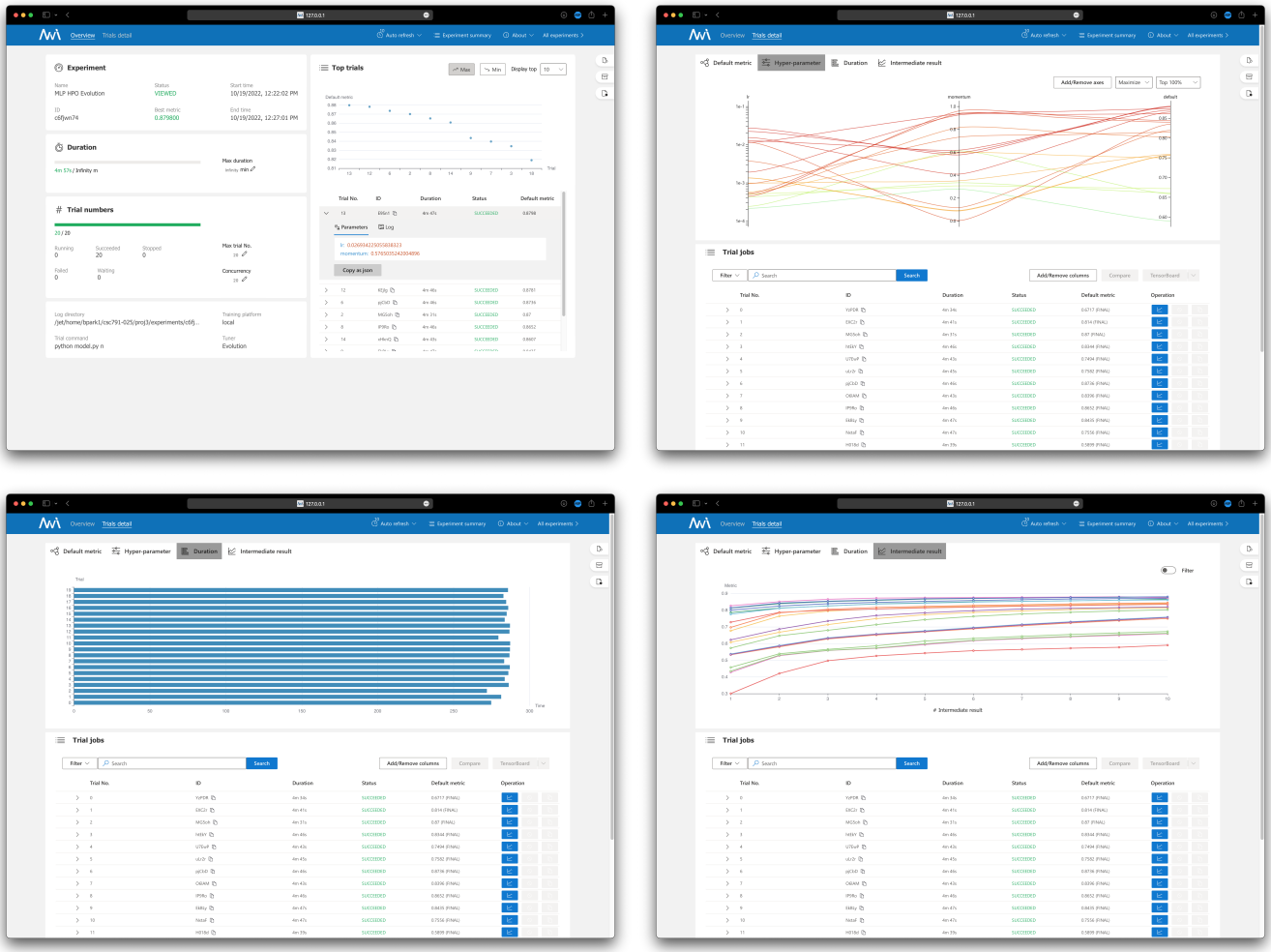

Figure 2: MLP with Evolution Tuner on Learning Rate and Momentum

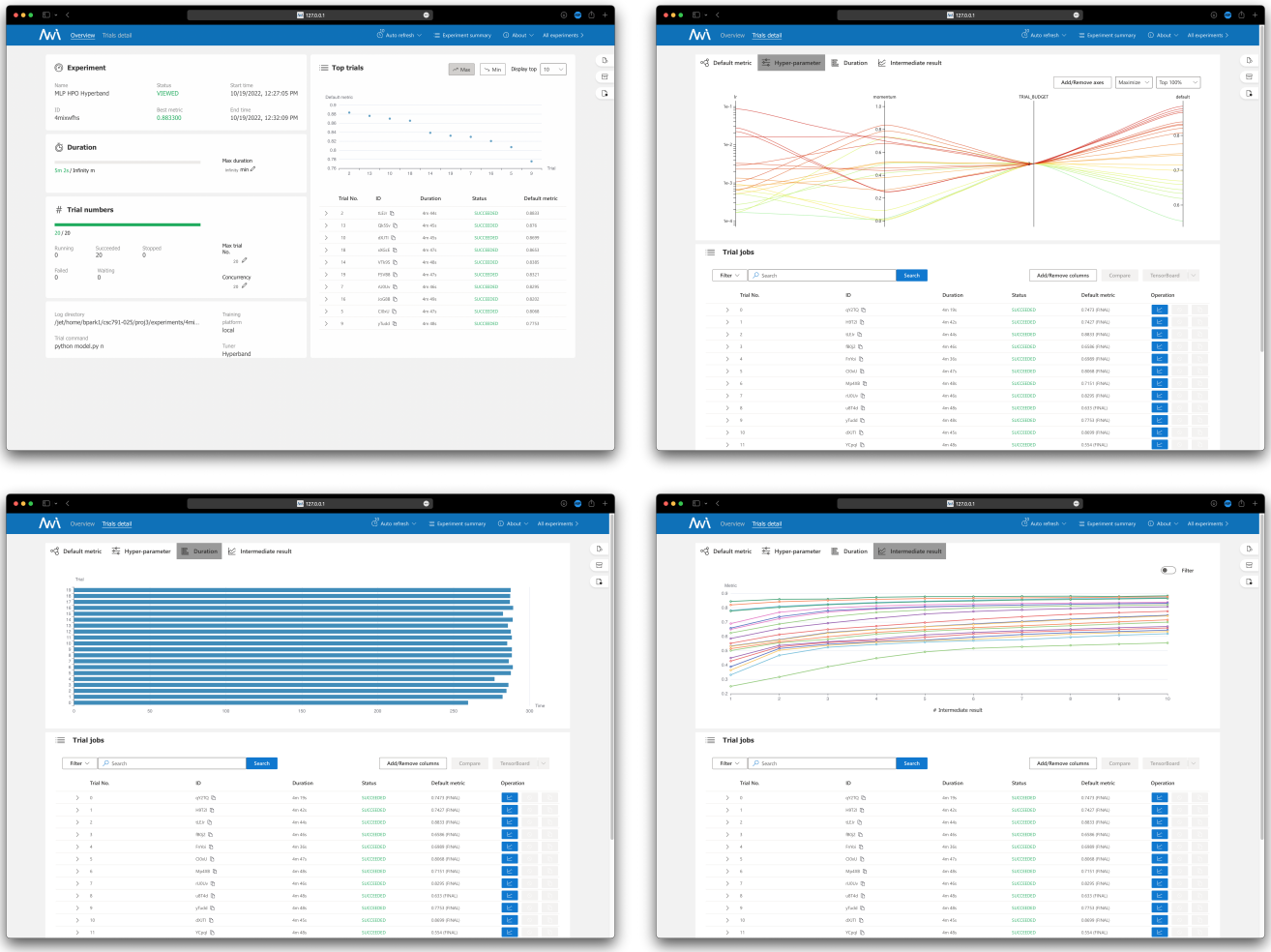

Figure 3: MLP with Hyperband Tuner on Learning Rate and Momentum

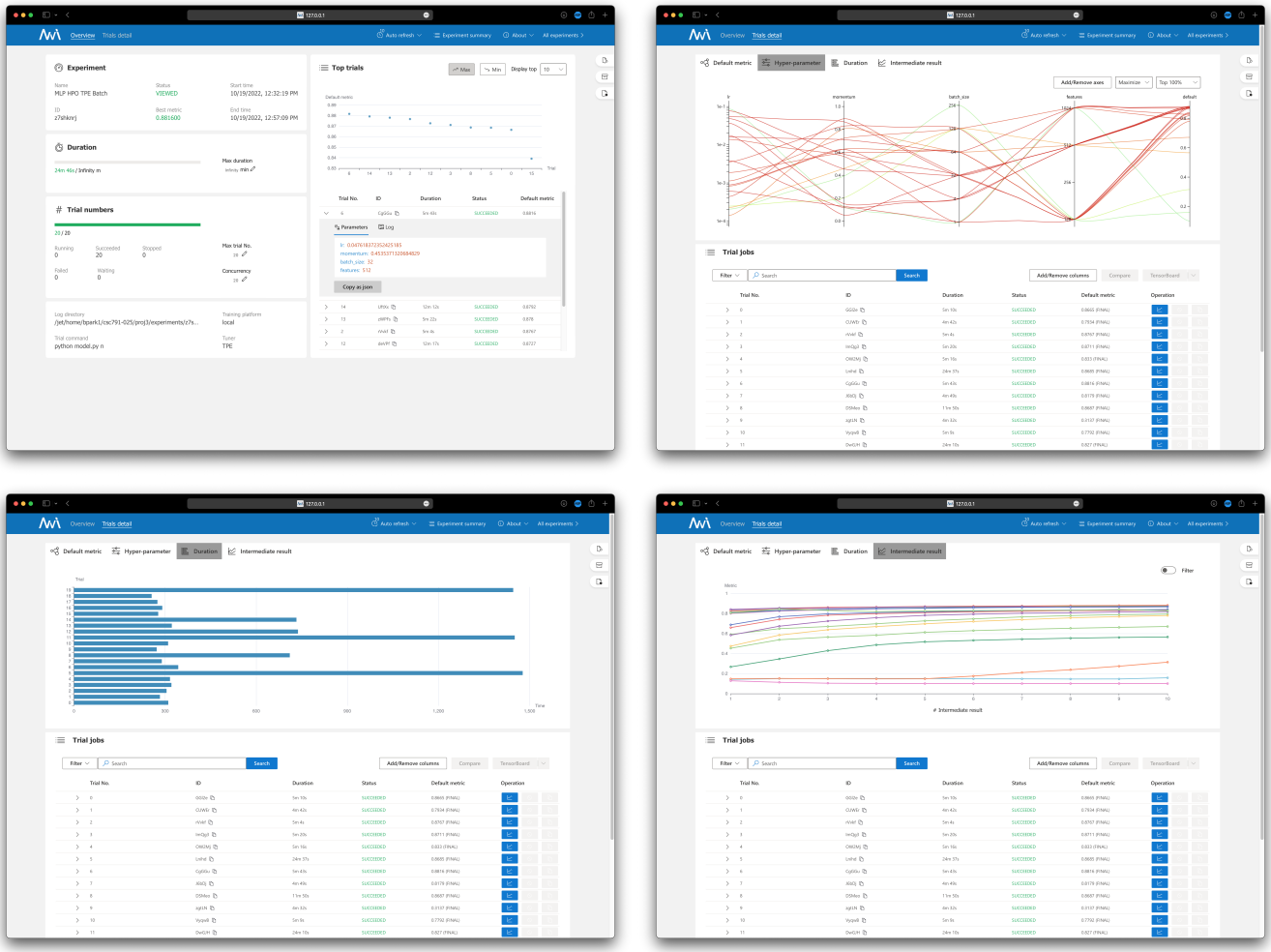

Figure 4: MLP with TPE Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

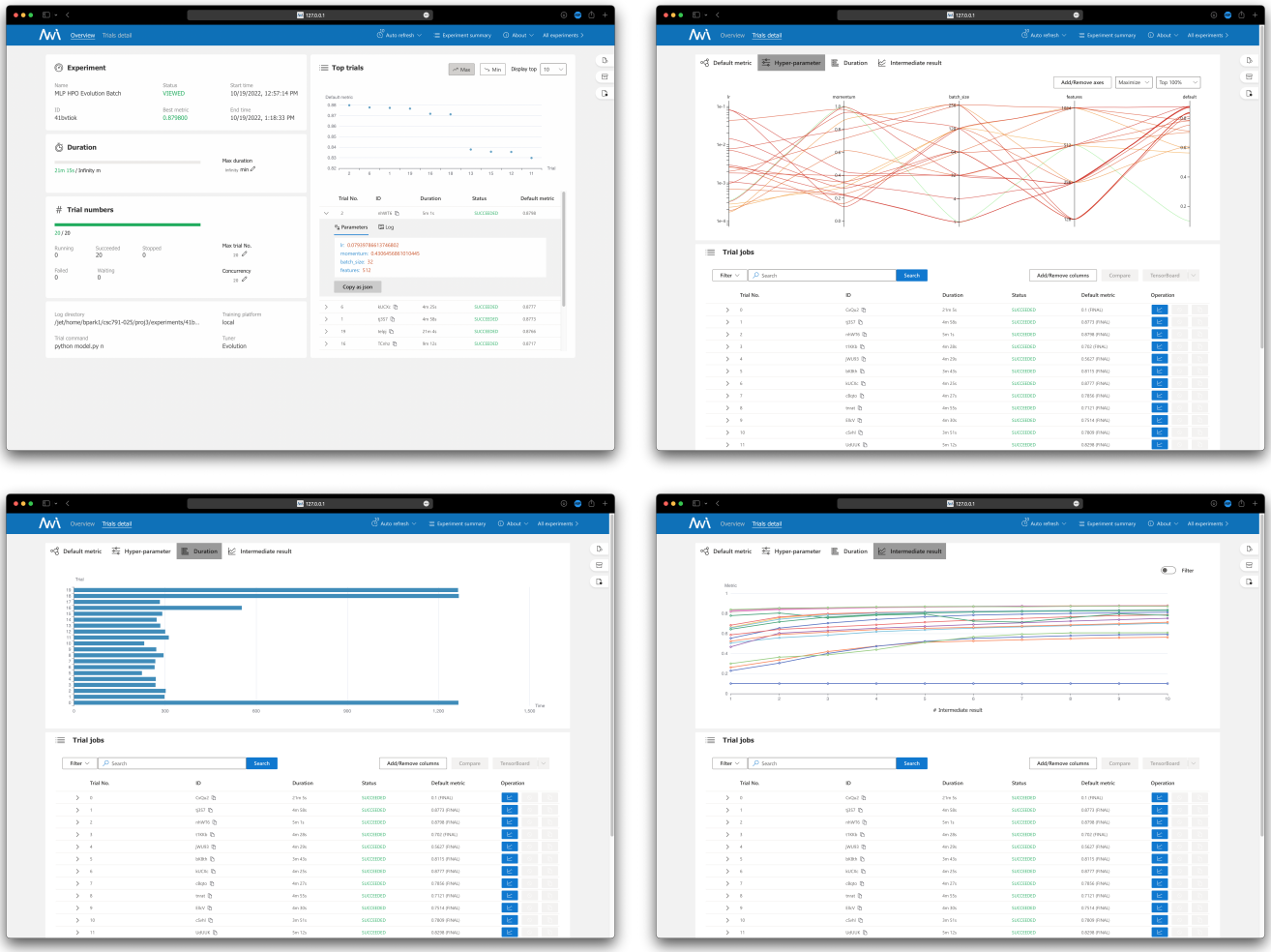

Figure 5: MLP with Evolution Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

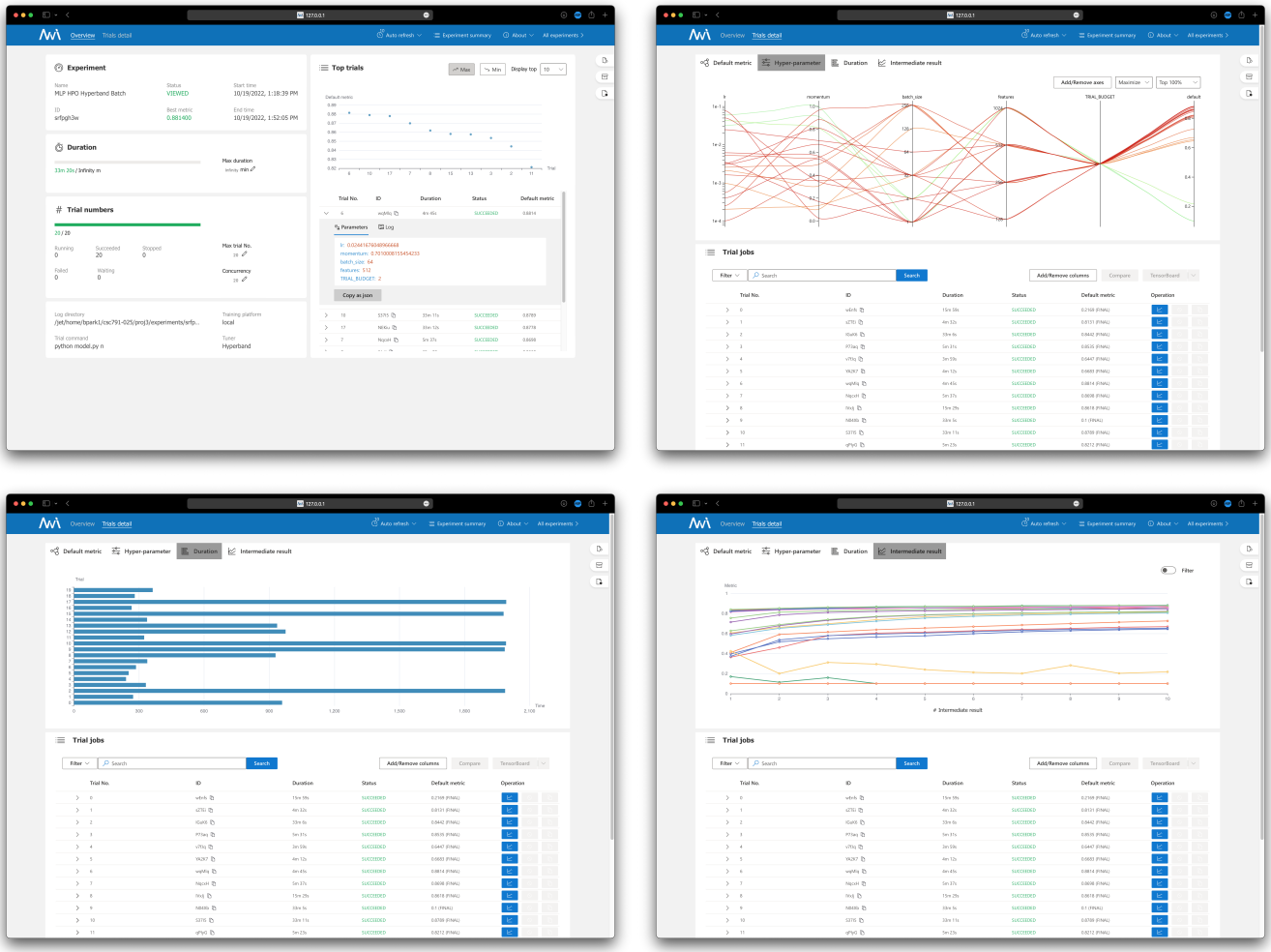

Figure 6: MLP with Hyperband Tuner on Learning Rate, Momentum, Feature Size, and Batch Size

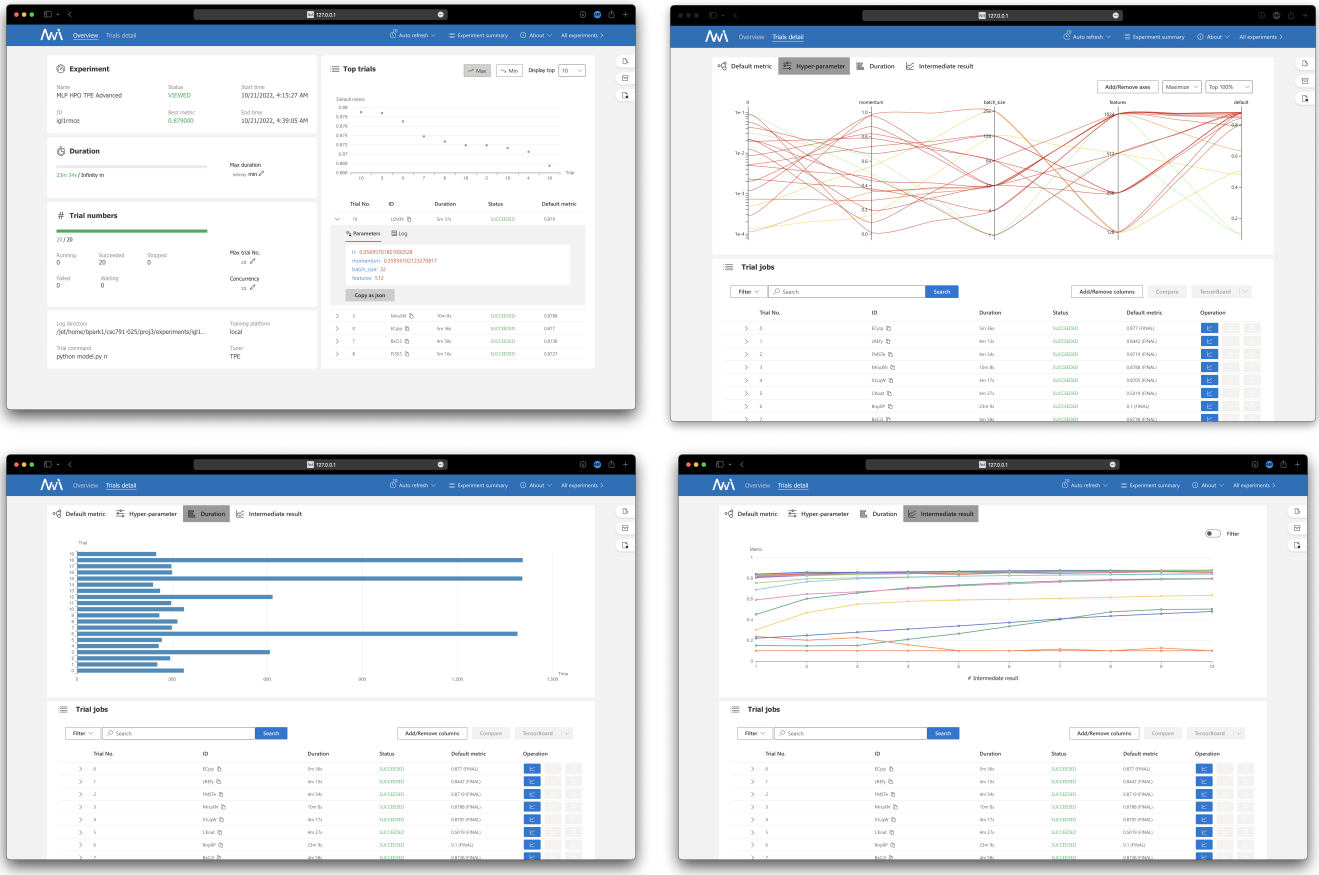

Figure 7: MLP with TPE Tuner on Learning Rate, Momentum, Feature Size, and Batch Size and Advanced HPO Configurations

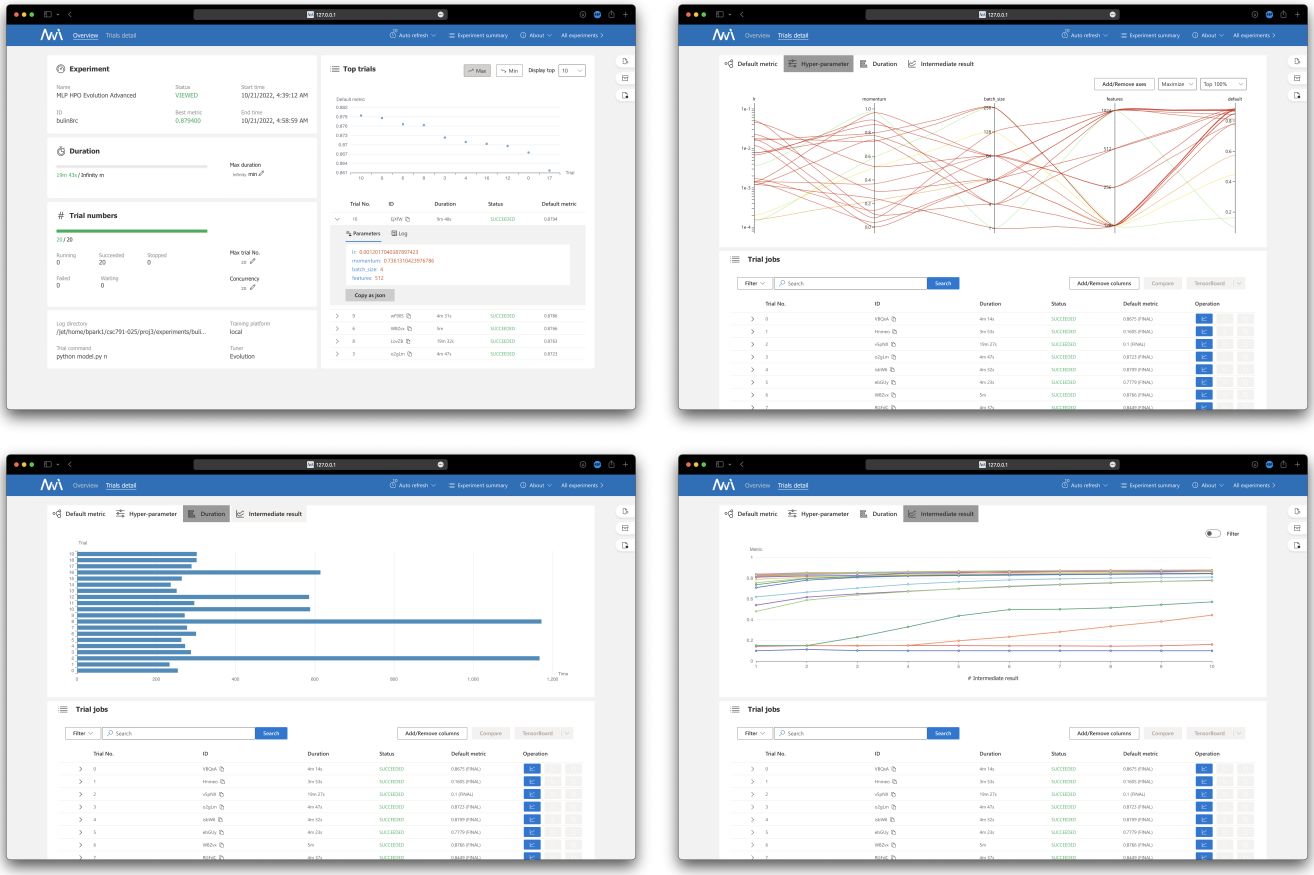

Figure 8: MLP with Evolution Tuner on Learning Rate, Momentum, Feature Size, and Batch Size and Advanced HPO Configurations

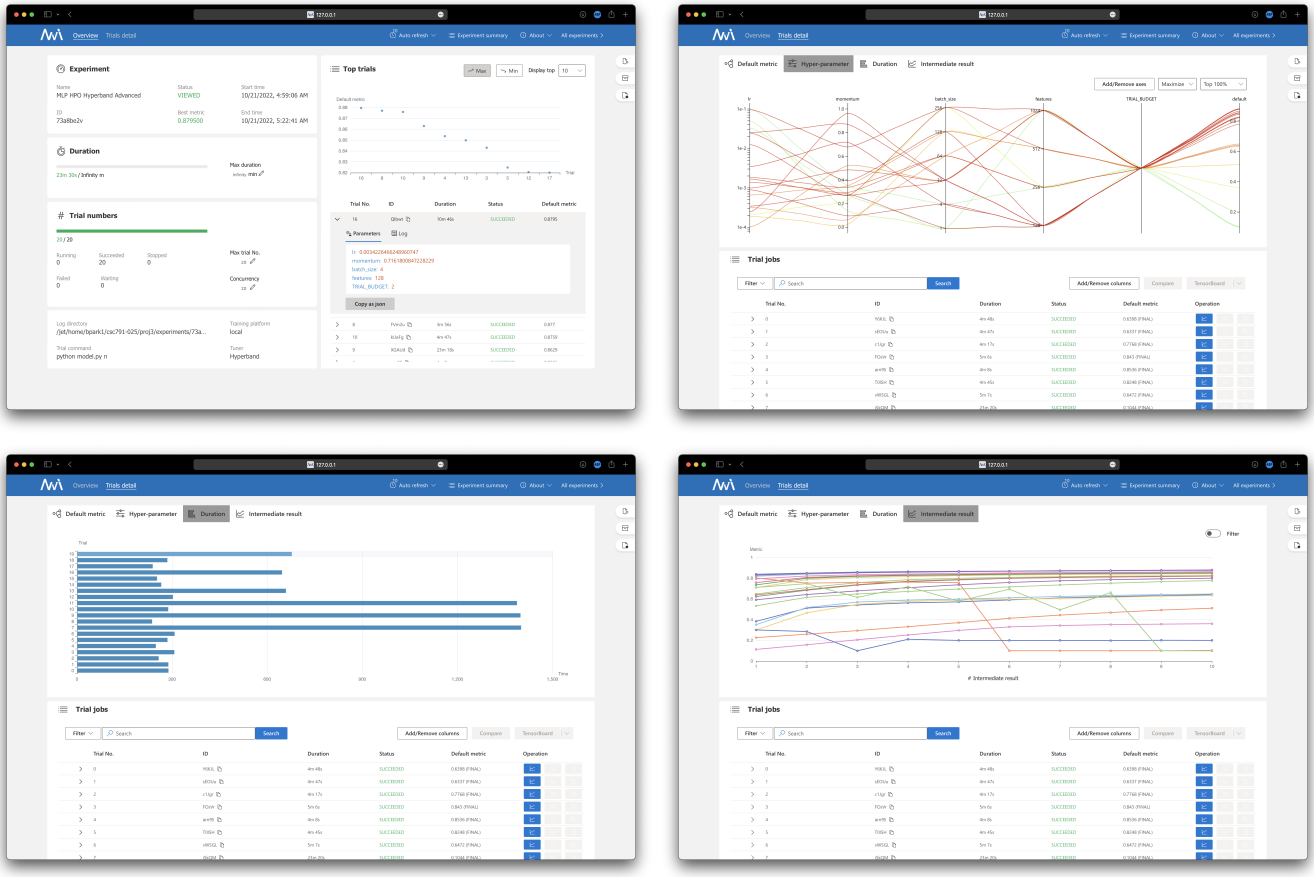

Figure 9: MLP with Hyperband Tuner on Learning Rate, Momentum, Feature Size, and Batch Size and Advanced HPO Configurations

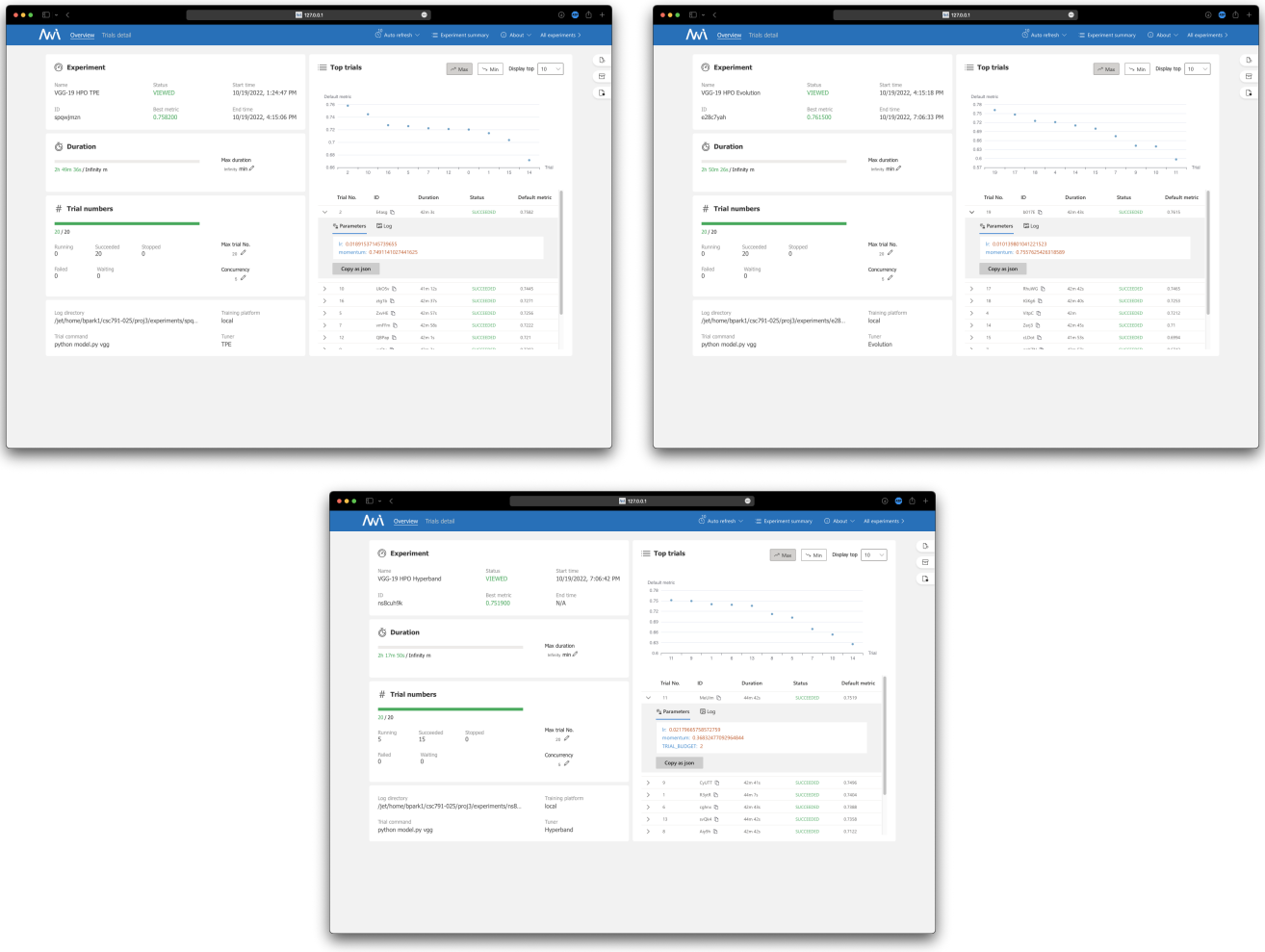

Figure 10: VGG HPOs on Learning Rate and Momentum