Записки научных
семинаров ПОМИ
Том 552, 2026 г.

N. Butakov, A. Vakhrushev, M. Khodorchenko, A. Zakharova, A. Ryzhkov, O. Plosskaya, A. Gainetdinov, R. Damdinov, D. Alexandrov, D. Nasonov, M. Savchenko

EFFICIENT DEADLINE-CONSTRAINED SCHEDULING OF DISTRIBUTED AUTOML PIPELINES

Abstract. Updating ML solutions in production on a daily basis is a common practice nowadays. Building new versions entails not only having automated ML (AutoML) pipelines at the ready for immediate data updates but also demands timely delivery, often within strict deadlines spanning only a few hours. To cope with such constraints, integrating timers into AutoML solutions is crucial, as it enables the allocation of time among multiple ML models, allowing for the exploration of different parameters to ensure good quality. This presents challenges in deciding how pipelines should compute to attain optimal performance. Scalable distributed frameworks such as Apache Spark when used as a base for AutoML frameworks may provide capabilities for blending data parallel and compute parallel modes into a hybrid mode that may deliver better overall performance under deadline constraints. However, setting parallelism parameters for multiple components with different scalability behavior may not be an easy task requiring to account for infrastructure, dataset volume, parameters of ML model and its scalability law. In light of these challenges, this paper presents a novel method for optimizing parallelism-related AutoML pipeline parameters and time-sharing amongst multiple ML models, all within the confines of a deadline constraint.

§1. Introduction

Modern organizations' business processes are heavily based on ML solutions of various sorts. These solutions often consist of multiple ensembled models combined and trained with appropriate hyperparameters using AutoML. Moreover, it often requires frequent updates on new incoming data on a daily basis. It creates a necessity to develop up-to-date solutions under strict deadlines, sometimes tightened in time by pre-processing, CI/CD, deployment or other processes requiring their share of time.

Key words and phrases: autoML and deadline-based scheduling and distributed system.

Such conditions require integrating timers into AutoML solutions, creating pressure on what and how they should be computed to achieve good enough performance. One of the approaches is to utilize cluster capabilities to the full extent.

Machine learning automation is a fairly extensive topic that has gained a lot of attention from scientific and technical communities under multiple aspects and perspectives: data types, industries, pipeline structures and CASH problems [3, 9, 11, 12, 15, 17, 18]. Scalable distributed AutoML solutions incorporate *compute* and *data parallel* approaches, combining them and focusing on different pipeline steps, e.g. automated hyperparameter tuning techniques, such as grid search, random search and hyperband optimization [1]. Though, mainly efforts in the subfield converge to providing a parameter, which allows the user to manually specify the number of models that should be trained simultaneously [9]. The other branch is frameworks employing hybrid parallel to optimize the runtime of ML jobs execution in distributed environments by offering a cost model (function) [4, 5, 14] or black box AutoML parameter optimization libraries [13]. In the paper [6], authors are focused on hybrid approaches and optimization using runtime prediction techniques made by regression models for different ML tasks [7]. Though, to the best of our knowledge, such works do not consider deadlines constraints and timer policies, which we deal with in this paper.

Choosing the right parallelism degree under a deadline is not easy because the model's performance may be affected by multiple factors that need to be considered to ensure adequate numbers. The situation gets even more complex if one needs to set parallelism degrees for multiple models possessing different performance traits simultaneously to execute them in the same AutoML pipeline.

In this paper, we propose a new method specially designed for and dedicated to optimizing AutoML pipelines parameters related to the parallelism of computing its inner ML models in hybrid mode on distributed data processing frameworks such as Apache Spark. The method takes into account deadline constraints and the policy of timer splitting and time allocating between individual parts of an AutoML pipeline.

§2. Deadline-constrained scheduling of AutoML pipelines

2.1. Predicting ML models execution time. In our paper, we show that selecting an appropriate configuration for the distributed environment

is a crucial aspect that impacts both costs of training and model accuracy. However, brute force exploration of all possible configurations does not make sense, as our objective is to enhance the speed of each AutoML run. To address this challenge, we propose using machine learning models to estimate the performance of different configurations instead of conducting honest AutoML evaluations. The overall runtime of each model in the pipeline consists of two primary components: 1) the training time of the algorithm on a single machine and 2) its scalability across the cluster.

In our example, we used an AutoML framework that specifically incorporated two types of ML algorithms: Gradient Boosted Decision Trees (GBDT) using LightGBM [10] implementation and the Linear Models by Spark MLLib. We focused only on binary classification and regression tasks for the experiment. As a result, we constructed the following two types of models:

- *Standalone* models were designed to estimate the training time of the GBDT and Linear models. The challenge in estimating the training time is that the system configuration will probably affect the result. To address this, our experiments involved estimating the training time in conventional units instead of seconds. We define a single unit as the average time it takes to perform a 10k x 10k matrix multiplication on itself, and obviously, it differs for every machine.
- *Scalability* models aim to estimate the scalability multiplier. It will predict how the evaluation time will change in distributed settings compared with standalone based on the number of executors provided.

To train the *Standalone* models, we collected two meta-datasets that contain the information about GBDT and Linear models runtime with different hyperparameters on both real-world datasets taken from OpenML [16] repository and synthetic datasets generated using the Guyon method [8]. The datasets varied in size, ranging from 20k to 10M rows and 10 to 2000 columns. Each meta-dataset contained dataset metadata, algorithm hyperparameters, the number of cores used during training, and the target variable representing the actual training time of the algorithm. Finally, we collected 2000 objects for Linear Models and 1342 for GBDTs.

We used the Random Forest regression model as a training time estimator, which was trained on a meta-dataset. For hyperparameter tuning and

Type of Model	$N_{samples}$	R^2 score
Standalone Linear	2000	0.663
Standalone GBDT	1342	0.412
Scalability Linear	55	0.93
Scalability GBDT	55	0.95

Table 1. CV results of meta modelling.

model evaluation of the Random Forest model, we employed group-5-fold cross-validation based on the dataset ID.

To train the *Scalability* models, we collected two meta-datasets consisting of 55 points each. Lager average dataset size ranging from 100k to 100M rows and increased computation costs substantiates the small size of the meta-dataset. Because of the tiny train sample size, we simplify the model structure and define it as follows:

$$\begin{aligned} t_{\text{train\_distr}} = t_{\text{train\_single}} * \big(w_0 + w_1/\log_{1.2} N_{\text{rows}} \\ + (w_2 + w_3/\log_{1.2} N_{\text{rows}})/\log_{1.2} N_{\text{exec}}\big), \end{aligned} \tag{1}$$

where w_i are parameters, N_{rows} is the number of rows in the training dataset, N_{exec} is the number of executors of Apache Spark, $t_{\text{train\_single}}$ is the result of training time on the single machine which is the result of *Standalone*, and $t_{\text{train\_distr}}$ is the final result in a distributed setup. These models were evaluated via leave-group-out cross-validation based on dataset ID.

The R^2 score was chosen as the performance metric. The summarized model performance results are presented in Table 1 and Figures 1 and 2. Meta-datasets creation code and model evaluation results are available in the repository[1].

2.2. AutoML pipeline modeling. To model an entire AutoML pipeline with several algorithms (see example in Figure 3), we should define the following: dataset reading time; preprocessing time; shuffle time to redistribute the data between executors, model hyperparameters tuning with Optuna algorithm and cross-validation; and the final solution ensembling, e.g. blending.

The overall scheduling procedure consists of infrastructure parameters selection defined by: n^i_{exec} – number of total executors per algorithm, n^i_{cores}

[1]The link will be available after the review

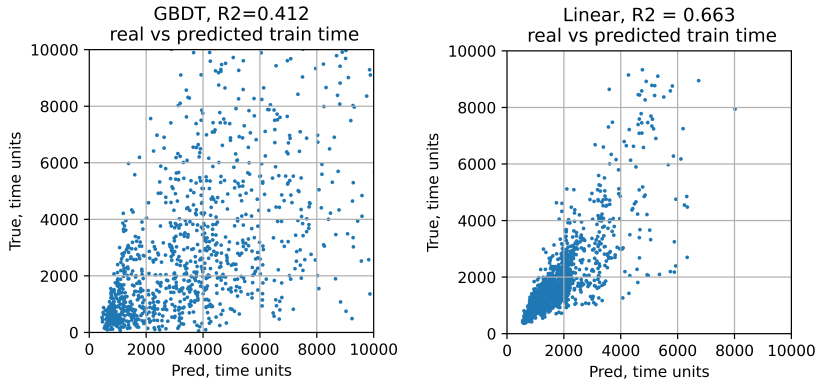

Figure 1. Standalone model performance scatter plot.

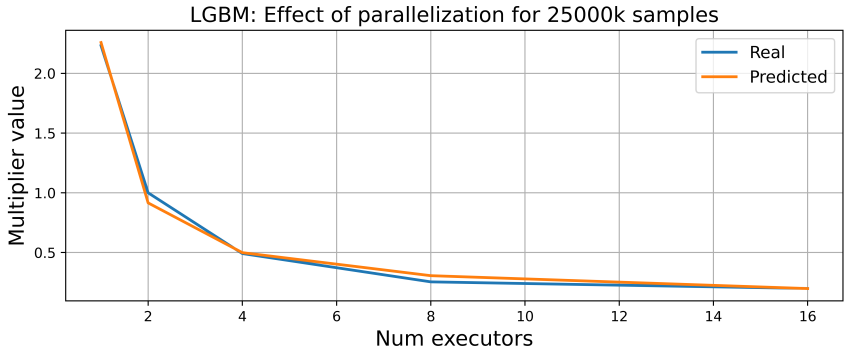

Figure 2. Scalability model real vs predicted curves.

– number of cores per executor, $n^i_{\text{core\_mem}}$ – the amount of memory per core in gb, $N^i_{\text{exec\_per\_trial}}$ – number of executors per trial, where a trial is an algorithm run.

Shuffle time t_{shuffle} is modelled by linear dependencies $a + b * N_{\text{exec}}$ with coefficients dependent on the total size of the dataset defined by $N_{\text{rows}} \times N_{\text{cols}}$. Dataset reading time and preprocessing time t_{preproc} were fitted to the following dependency $1/N_{\text{exec}} * a + c$. All the corresponding coefficients were learned from actual runs.

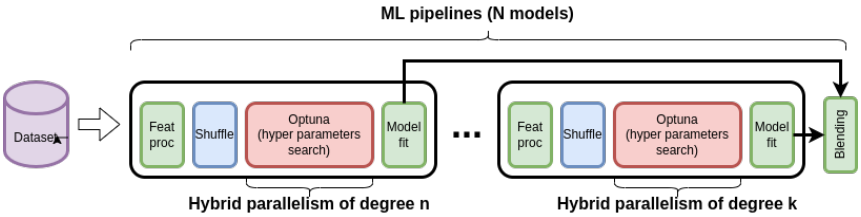

Figure 3. AutoML pipeline consisting of several ML models, tuned in parallel mode with Optuna and blended into ensemble.

The overall algorithm running time includes operations that are made once and

$$t^i_{\text{pipe}} = f_{\text{shuffle}}(N^i_{\text{exec}}, N^i_{\text{exec\_per\_trial}}, D_{\text{size}}) + f_{\text{read}}(N_{\text{rows}}, N_{\text{cols}}, N^i_{\text{exec}}) + f_{\text{preproc}}(N_{\text{rows}}, N_{\text{cols}}, N^i_{\text{exec}}) + f_{\text{opt}}(t^i_{\text{train\_distr}}, N^i_{\text{exec}}, N^i_{\text{exec\_per\_trial}}, \Psi) + t^i_{cv}, \quad (2)$$

where i is an algorithm from the predefined set of algorithms in AutoML pipeline A, f_{opt} – optimization time function and Ψ – time or iteration constraint depends on a strategy and is discussed in section 3.3.

The overall pipeline time can be formulated as:

$$T_{\text{pipe}} = \sum_{i \in A} t^i + t_{\text{ens}} \quad (3)$$

with the following objectives:

$$Q_{\text{potential}} = \sum_{i \in A} \log(N^i_{\text{algo\_trials}}) \quad (4)$$

$$Q_{\text{infrastructure}} = \sum_{i \in A} (C_{\text{exec}} \cdot N^i_{\text{exec}} + C_{\text{core}} \cdot N^i_{\text{core}} + C_{\text{core\_mem}} \cdot N_{\text{core\_mem}}) \cdot t^i, \quad (5)$$

where C_{exec}, C_{core} and $C_{\text{core\_mem}}$ are constants to the model executor, core and memory per core costs, respectively.

We allow making single or multiobjective optimization with two quality functions $Q_{\text{potential}} \to \max$ and $Q_{\text{infrastructure}} \to \min$ with the memory constrain $N^i_{\text{cores}} \cdot N^i_{\text{exec\_per\_trial}} \cdot N^i_{\text{core\_mem}} >= D_{\text{vol}}$, where D_{vol} is volume of the full dataset in GB.

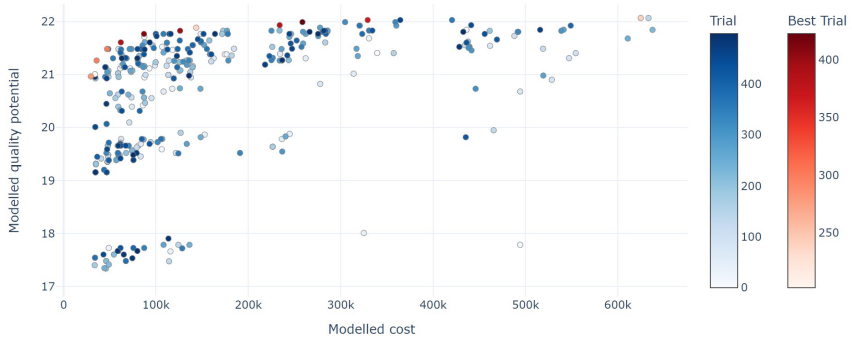

Figure 4. Pareto-front of solutions for 1M dataset case with deadline of 3 hours for AutoML pipeline with 3 algorithms and near-equal time strategy.

2.3. Deadline-constrained optimizer. The effectiveness of scheduling is highly dependent on the computation deadlines, which can be dictated by the cost of infrastructure, fast-prototyping needs or limited time for models update. Thus, we must follow the deadline constraint: $T_{\text{pipe}} \leq$ Deadline.

We can redistribute calculation time between the algorithms to be tuned according to the deadline. This time is defined as the overall deadline time without overheads, i.e., pure f_{opt}. We define two main strategies here: 1) near-equal time when all t^i_{opt} are considered to be equal, allowing to provide different iteration numbers for algorithms which can be estimated as

$$t^i_{\text{opt}} = \text{Deadline}^i \Big(\mod \big(\frac{N^i_{\text{exec}}}{N^i_{\text{exec\_per\_trial}}} \cdot t^i_{\text{train\_distr}}\big)\Big) \cdot t^i_{\text{train\_distr}}, \quad (6)$$

and 2) near-equal iterations, when there is no evident preference for each of the algorithms, and the goal is to ensure diversity of solutions. The latter case is formalized by a set of linear equations, which are aimed at finding the closest number of iterations in cases when there is no user-defined apriory advantage between algorithms. In both cases it is essential to select a good parallelization parameters to ensure good performance and quality.

Dataset	N_{rows}	N_{cols}	Target
used_cars	3000040	65	regression
synth_5kk_100	5000000	100	binary
synth_10kk_100	10000000	100	binary
kaggle_a	21198036	61	binary

Table 2. Experimental datasets.

§3. Experimental study

3.1. Experimental setup. In order to verify the proposed approach, we conducted a set of experiments on several selected datasets of different sizes, numbers of columns and target (see Table 3.1).

We took LightGBM and Linear ML algorithms and calculated up to 64 Optuna [2] trials for them on all the abovementioned datasets. For each algorithm/dataset pair, we employed a distributed Spark application consisting of 16 executors and calculated the series of trials several times using different parallelism degree ranging from 1 (pure *data parallel* mode) to 16 (pure *compute parallel* mode). Each parallelism degree is required to perform a shuffle to prepare the desired number of independent datasets copies spanning a subset of executors in the app (also ranging from 1 to 16). That was necessary to ensure the proper functioning of hybrid data/compute parallel runs simultaneously and correctly consider all induced overheads. All computations were conducted on Kubernetes cluster of 8 worker nodes with 16 CPU and 128 GB RAM.

Some of the series are presented on Figures 5, 6, 7. For each trial, we registered start and end times.

Hyperparameter search with Optuna generates configurations for the algorithms, thus adding to the variance of its execution times alongside the natural stochasticity of the computational environment and the training process itself. That contributes to uneven start and end times of trials independent of chosen parallelism degree and may eventually lead to violating the schedule.

Using the series and setting hypothetical deadlines in the middle of them, we could model various situations with partially calculated series and thus compare how many trials can be finished before the deadlines for every parallelism degree under consideration.

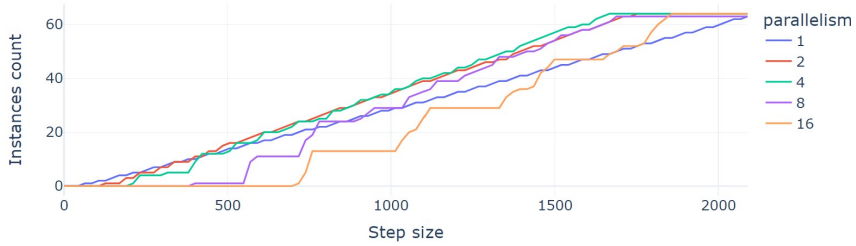

Figure 5. Execution time of linear algorithm on kaggle_a with different parallelism ranging from 1 to 16 (available 16 executors with 4 cores in all runs).

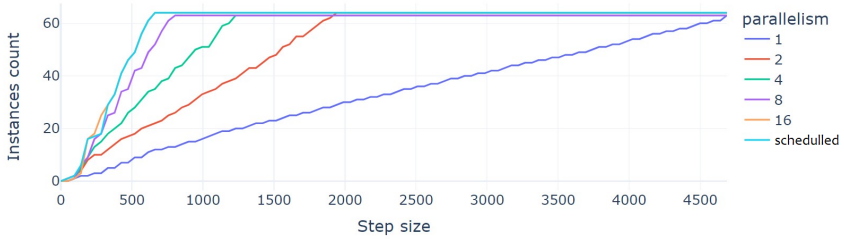

Figure 6. Execution time of lightgbm algorithm on used_cars_dataset_1x with different parallelism ranging from 1 to 16 (available 16 executors with 4 cores in all runs).

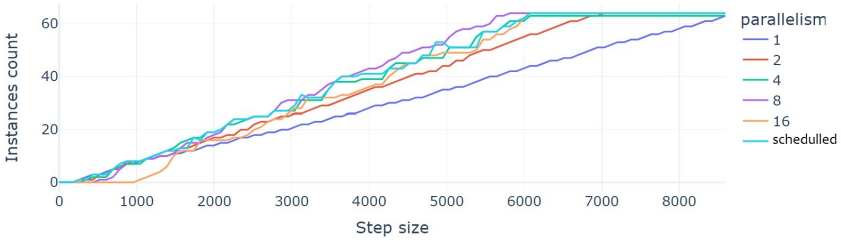

Figure 7. Execution time of lightgbm algorithm on kaggle_a dataset with different parallelism ranging from 1 to 16 (available 16 executors with 4 cores in all runs).

Each point of these series depicted on Figures 5, 6, 7 tells how many instances we would be able to calculate with the corresponding parallelism degree up to the potential deadline matching the time of the point.

Using the proposed optimizer, we can also estimate the desired parallelism degree for all the deadlines presented with these points and thus trace our own series combined from points taken from different series matched by the chosen parallelism degree for each deadline. The resulting series are presented on Figures 6, 7.

We also explicitly compared our approach with a couple of natural baselines: (1) *data parallel* mode assuming parallelism degree of 1; (2) *compute parallel* mode assuming max affordable parallelism degree (if the dataset can coalesce into one executor, then the degree is equal to the number of executors in the app, e.g. 16 or 32 in our experiments). Tables 3.2 and 3.2 show a reduction in the time compared to *data parallel* and *compute parallel* modes, respectively, for achieving the same number of finished trials.

3.2. Discussion. Figures 5, 6, 7 demonstrates that optimal parallelism degree may be affected by not only the scalability law of the model but also the relation of overheads to model's compute time (Figure 5, higher parallelism degrees are less beneficial due to high overheads on shuffling). Dataset size and its relation to available infrastructure is a dominant factor (on par with a scalability law of the model) that may render some parallelism modes (Figure 7, low degree modes are significantly less efficient) or all of them except one (Figure 6 compute parallel mode is the winner for all deadlines) inefficient.

The latter fact brings attention to flexible adjusting of parallelism: on Figure 7 one may see that for 10 trials or less, it is better to use a degree of 2 or 4, while for a greater number of trials, 16 is the best. Such adjustments may yield 10-20% more finished trials.

The series obtained with the proposed optimizer (examples in Figures 6 and 7), though not dominant for each particular case, demonstrates significant improvement to one or both baselines. It allows removing the burden of choosing an efficient parallelism degree off the user's shoulders by leveraging automatic choice with the optimizer.

Results presented in Tables 3.2 and 3.2 show improvement obtained by applying the optimizer to LightGBM and Linear ML algorithms. In most cases, the approach shows better performance by accounting for overheads induced by data transferring and choosing the right degree of parallelism.

Dataset	Nex	5	10	20	30	40	50	60
kaggle_a	16	147,-7	49,3	37,33	0,25	21,47	20,43	8,42
kaggle_a	32	151,-11	39,-11	24,60	0,55	22,79	4,87	0,110
synth_10kk _100	16	62,-1	32,45	9,84	0,107	3,128	-6,135	0,149
synth_5kk _100	16	80,64	0,63	-7,142	0,273	-14,216	-20,213	0,336
used_cars _dataset_1x	16	35,114	0,160	-6,311	0,442	0,531	0,552	0,634

Table 3. Improvement of lgb algorithm trials execution time (in %) obtained with the hybrid approach in comparison to *data parallel* / *compute parallel* configuration (values splitted by a comma).

Dataset	Nex	5	10	20	30	40	50	60
kaggle_a	16	102,18	26,34	33,41	0,43	12,46	6,37	-3,36
synth_10kk _100	16	45,0	14,34	23,48	0,57	9,55	-4,45	0,54
synth_5kk _100	16	57,38	6,47	9,64	0,86	6,83	-6,74	0,83
used_cars _dataset_1x	16	46,300	0,435	-51,243	0,563	0,456	0,511	0,518

Table 4. Improvement of linear algorithm trials execution time (in %) obtained with the hybrid approach in comparison to *data parallel* / *compute parallel* configuration (values splitted by a comma).

It tends to *data parallel* mode for a low number of trials, while for high numbers of trials the approach goes closer to *compute parallel*.

§4. Conclusion

In this paper, a new method for optimization of AutoML pipelines parallelism parameters has been proposed. For high number of trials in tuned ML models, the method may yield better results for up 20–40% better than pure *data parallel* or *compute parallel* mode exploiting hybrid mode and considering deadline constraints. The approach may be further developed

by incorporating meta-learning techniques for quality curves identification and taking them into consideration to set appropriate coefficients for timer splitting between ML models in the AutoML pipeline.

References

1. A. Abd Elrahman, M. El Helw, R. Elshawi, and S. Sakr, *D-SmartML: A Distributed Automated Machine Learning Framework*, in: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), 2020, pp. 1215–1218.
2. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A Next-Generation Hyperparameter Optimization Framework*, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.
3. M. Bahri, F. Salutari, A. Putina, and M. Sozio, *AutoML: State of the Art with a Focus on Anomaly Detection, Challenges, and Research Directions*, International Journal of Data Science and Analytics **14** (2022), no. 2, 113–126.
4. M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthör, K. Innerebner, F. Klezin, S. Lindstaedt, A. Phani, B. Rath, *et al.*, *SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle*, ArXiv preprint arXiv:1909.02976 (2019).
5. M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, *et al.*, *SystemML: Declarative Machine Learning on Spark*, Proceedings of the VLDB Endowment **9** (2016), no. 13, 1425–1436.
6. M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. R. Burdick, and S. Vaithyanathan, *Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML*, Proceedings of the VLDB Endowment **7** (2014), no. 7, 553–564.
7. P. Dube, T. Salonidis, P. Ram, and A. Verma, *Runtime Prediction of Machine Learning Algorithms in AutoML Systems*, in: ICASSP 2023 – 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2023, pp. 1–5.
8. I. Guyon, S. Gunn, A. B. Hur, and G. Dror, *Design and Analysis of the NIPS2003 Challenge*, in: Feature Extraction: Foundations and Applications, 2006, pp. 237–263.
9. X. He, K. Zhao, and X. Chu, *AutoML: A Survey of the State-of-the-Art*, Knowledge-Based Systems **212** (2021), 106622.
10. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, in: Advances in Neural Information Processing Systems **30**, 2017.
11. T. T. Le, W. Fu, and J. H. Moore, *Scaling Tree-Based Automated Machine Learning to Biomedical Big Data with a Feature Set Selector*, Bioinformatics **36** (2020), no. 1, 250–256.
12. X. Shi, J. Mueller, N. Erickson, M. Li, and A. Smola, *Multimodal AutoML on Structured Tables with Text Fields*, in: 8th ICML Workshop on Automated Machine Learning (AutoML), 2021.

13. X. Song, S. Perel, C. Lee, G. Kochanski, and D. Golovin, *Open Source Vizier: Distributed Infrastructure and API for Reliable and Flexible Blackbox Optimization*, in: International Conference on Automated Machine Learning, 2022.
14. E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht, *KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics*, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), 2017, pp. 535–546.
15. A. Vakhrushev, A. Ryzhkov, M. Savchenko, D. Simakov, R. Damdinov, and A. Tuzhilin, *LightAutoML: AutoML Solution for a Large Financial Services Ecosystem*, ArXiv preprint arXiv:2109.01528 (2021).
16. J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo, *OpenML: Networked Science in Machine Learning*, ACM SIGKDD Explorations Newsletter **15** (2014), no. 2, 49–60.
17. A. Visheratin, A. Struckov, S. Yufa, A. Muratov, D. Nasonov, N. Butakov, Y. Kuznetsov, and M. May, *Peregreen – Modular Database for Efficient Storage of Historical Time Series in Cloud Environments*, in: 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020, pp. 589–601.
18. M.-A. Zöller and M. F. Huber, *Benchmark and Survey of Automated Machine Learning Frameworks*, Journal of Artificial Intelligence Research **70** (2021), 409–472.

ITMO University, St.Petersburg, Russia
E-mail: alipoov.nb@gmail.com

Sber AI Lab, Moscow, Russia
E-mail: btbpanda@gmail.com

ITMO University, St.Petersburg, Russia
E-mail: mariyaxod@yandex.ru
E-mail: nastyazakharova.nz@gmail.com

Sber AI Lab, Moscow, Russia
E-mail: alexmryzhkov@gmail.com
E-mail: nonflame@gmail.com

ITMO University, Saint-Petersburg, Russia
E-mail: Mr.g.azamat@gmail.com

Sber AI Lab, Moscow, Russia
E-mail: damdinovr@gmail.com

ITMO University, Saint-Petersburg, Russia
E-mail: mr.alexdmitriy@mail.ru
E-mail: denis.nasonov@gmail.com

Sber AI Lab, Moscow, Russia
E-mail: savvvan@gmail.com

Поступило 4 февраля 2026 г.