DOI: 10.14529/jsfi230207

# Improving Efficiency of Hybrid HPC Systems Using a Multi-agent Scheduler and Machine Learning Methods

*Vladimir S. Zaborovsky*[1], *Lev V. Utkin*[1], *Vladimir A. Muliukha*[1], *Alexey A. Lukashin*[1]

One of the promising directions for improving hybrid reconfigurable high-performance computer platforms operating in the mode of collaborative applied computing centers is their inclusion as an active component in the machine learning ecosystem, which opens up new opportunities to enhance the actual outperformance of solving various application tasks by intellectualizing the management of available computing resources. The task scheduler operation is crucial in improving the efficiency of hybrid supercomputer platforms, which combine dozens of processor blocks with different architectures, including specialized graphics and reconfigurable accelerators. To form an optimal order of jobs in the HPC queue, the article proposes to apply deep survival machine learning models, which increase the accuracy of the estimated time of the tasks successful execution and the required amount of computing resources. The main peculiarity of the machine learning models is that they are trained on censored heterogeneous data collected from previous periods of task execution observations using a multi-agent scheduler. In order to ensure high accuracy, the random survival forest is used as a part of the machine learning model which provides survival and hazard functions in the framework of the survival analysis. A specific weighted clustering procedure is proposed to divide tasks in accordance with their execution times as well as the feature vectors. Various numerical experiments with actual data illustrate the outperformance of the presented approach.

*Keywords: high performance computing, hybrid computing systems, machine learning, multi-agent scheduler, random survival forest, survival analysis, survival function, XAI.*

## Introduction

A modern supercomputer is a very complex technical system that simultaneously performs quadrillions (a number with 15 zeros) operations used to solve complex mathematical computations and process huge amounts of data that arise in the process of solving various scientific and engineering problems. Supercomputer users, whose number is steadily growing every year, are interested in the real result of their calculations, and not just the peak performance of the supercomputer, which is nominally expressed in the number of floating-point arithmetic operations per second. As the performance of supercomputers increases, the complexity of their use also increases. Therefore, users need to have a deep knowledge not only in the field of their professional activity but also real skills in the most efficient use of available computing resources to consume the real performance of a supercomputer. In this process, the task scheduler plays an important role, combining different groups of processors, including graphics (vector) and reconfigurable (FPGA) accelerators, into a consistent hybrid computing field available for a specific user task. To improve the efficiency of the scheduler for a wide range of applied tasks, machine learning models with attention mechanisms are proposed. These models allow calculating with high accuracy the probability of successful completion of a user task in a given time interval, as well as evaluating specific task parameters that can greatly affect the actual performance of the hybrid platform. The system parameters, machine learning models, and promising transformer-based architecture presented in the article were obtained based on the analysis of the functioning of a

[1]Peter the Great St.Petersburg Polytechnic University, St.Petersburg, Russian Federation

hybrid supercomputer cluster of St. Petersburg Polytechnic University with a peak performance of more than 1 PFlops.

One of the tools for dealing with the task completion problem is survival analysis [7] which is used in many applied areas [13, 20, 27]. A comprehensive review of survival analysis methods and their implementation by the machine learning models can be found in [27]. An important peculiarity of the survival models is that their outcomes are functions (the survival function, the hazard function, the cumulative hazard function, etc.) as predictions instead of point-valued data which are predicted by most machine learning models.

All survival models can be divided into three groups. The first group consists of parametric models for which a probability distribution of time to event is known, but its parameters are unknown. The second group contains semi-parametric models for which it is assumed that a functional dependence between features and the model outcomes is known. The well-known Cox proportional hazards model [4] belongs to the second group. Models of the third group are called non-parametric. They assume that the probability time-to-event distribution is unknown, and the relationship between features and the model outcomes is also unknown. The Kaplan–Meier model [27] is one of the models from the third group.

Following the Cox model, many its extensions have been proposed, which use the non-linear relationship between covariates and the time of event [20]. A lot of models are based on neural networks. However, the difficulty to train a neural network, when the number of training examples is restricted, led to applying another approach based on using the random survival forests [2], which can be regarded as an extension of the original random forest [3] to the case of survival analysis. It turns out that the random survival forests are a powerful and efficient tool for survival analysis due to their several useful peculiarities. First of all, random forests require a few tuning parameters [9]. Random forests are highly data adaptive and can deal with both low and high dimensional data [26]. Therefore, we use the random survival forests as a basic model for analyzing the task completion time.

Our contributions can be summarized as follows:

1. Collection of job execution data and comprehensive statistical analysis of performed jobs taking into account domain and user characteristics.
2. Survival analysis for computing probabilistic characteristics of the task completion time (survival functions, expected times of the task completion, etc.) is proposed to be applied. One of the reasons for using survival analysis is the availability of many tasks which have been terminated due to several reasons and are considered as censored data.
3. The predictions of the task completion time by means of a set of the random survival forests are proposed to be performed. The random survival forest is a powerful and efficient tool for the case when the number of training data is limited.
4. A specific procedure for clustering training data and for training several random survival forests is proposed, which allows us to take into account the fact that tasks may be quite different and they should be separated into clusters with a homogeneous structure. This approach is supplemented by a specific procedure of computing predictions for new tasks.
5. A simple approach for taking into account the user history is proposed, which is reduced to computing probability distributions of different task completion events. The corresponding probability distributions are concatenated with the initial feature vector obtained for the analyzed user.

6. We consider questions how to explain prediction of the task completion time in order to have an opportunity for the user to change the task parameters or just to understand why the corresponding task can be completed unsuccessfully.
7. An approach of integrating the developed prediction model into the existing environment of a hybrid supercomputer environment by integration of the predicted execution time of a job to a scheduler.

The article is organized as follows. A description of the supercomputer analyzed and of its component (the Slurm Task Scheduler) is given in Section 1. General approaches to improving the efficiency of the SCC are discussed in Section 2. Elements of survival analysis are considered in Section 3. The main algorithm for the task completion prediction is provided in Section 4. Ways for improving the proposed algorithm based on survival analysis are described in Section 5. Section 6 considers the important question of interpretation of the machine learning analysis results. Numerical experiments with the analysis of results are considered in Section 7. Concluding remarks and perspectives are discussed in the Conclusion section.

## 1. Current Situation in the Supercomputer Center "Polytechnic"

### 1.1. Description of the Supercomputer Center "Polytechnic"

Supercomputing Centre "Polytechnic"(hereinafter SCC) is a hybrid high performance computing system consisting of several computing clusters with different architectures (homogeneous and heterogeneous) located in a single information and computing field and connected using 56 Gb/s Infiniband FDR, as well as a common storage Luster, with a volume of about 1 PB, which allows file exchange between different computing clusters. All SCC systems provide more than 1.5 PFlops performance on double-precision floating point computation and more than 2.5 PFlops on single and half-precision computations that are typically used for training machine learning algorithms.

All registered users get access to SCC resources from the dedicated server using the terminal client protocol. SCC resources are managed using the Slurm Workload Manager. The principle of its operation is as follows (Fig. 1): the user requests some resource (processor cores, memory, etc.), placing his task in the queue; the system, based on the user's priorities and the current filling of the queue, selects the moment of task launch. A queue is a sequence of tasks that must be solved on a specific computing resource (a group of nodes). At the same time, each node at the current time can be occupied by only one task of one user. Thus, the node is assigned to the exclusive use of the task hosted on it, and other tasks on the busy node will not be executed.

The life cycle of a task created by a user in the SCC consists of four stages:

- user connection to the console of the control node of the SCC through a client (for example, PuTTY for Windows OS, or built-in SSH client for Linux OS);
- creating and running a task in the console in interactive or batch mode;
- execution of the task on the selected resources of the SCC;
- completion of the task.

SCC provides different systems by dividing resources into multiple queues (partitions in SLURM terminology). Currently, the following clusters are available:

- Tornado: homogeneous cluster with 28-core and 64 GB of RAM nodes;

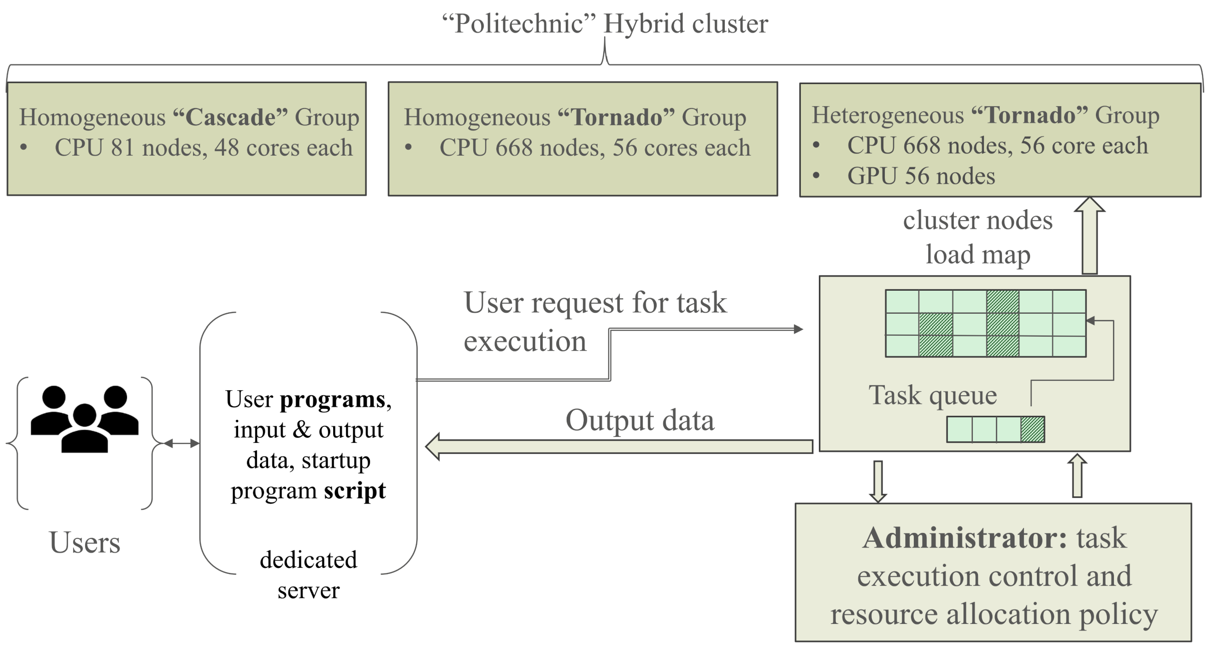

"Politechnic" Hybrid cluster
Homogeneous "Cascade" Group
• CPU 81 nodes, 48 cores each
Homogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
Heterogeneous "Tornado" Group
• CPU 668 nodes, 56 core each
• GPU 56 nodes
cluster nodes load map
User request for task execution
Task queue
Output data
User programs, input & output data, startup program script
dedicated server
Users
Administrator: task execution control and resource allocation policy

**Figure 1.** Structure and information flows of the supercomputer center "Polytechnic"

- Tornado-k40: heterogeneous cluster with 28-core, 64 GB of RAM, and 2 Tesla GPUs nodes;
- Cascade: homogeneous cluster with 48-core and 192 GB of RAM nodes;
- NV: heterogeneous cluster with 48-core, 768 GB of RAM, and 8 Tesla GPUs nodes.

Every job is submitted into a particular partition with specific resources available. All clusters are connected to the same storage. That provides users with the ability to use different compute resources for the different stages of numerical simulations and data processing.

## 1.2. Description of the Slurm Task Scheduler

Slurm is a cluster management and task planning system. It performs three main functions:

- distributes access to resources (computing nodes) for users;
- provides an environment for launching, executing and monitoring tasks on dedicated nodes;
- manages the task queue.

The user can submit tasks in two modes via the command line – interactive and batch modes.

In interactive mode, the user can act according to two algorithms. In the first case, resources are requested, the application necessary for operation is launched on the head node of the SCC, and then the user is working on the application. In the second case, a connection is made to the selected node, and the rest of the actions are performed on this node.

In batch mode, the user prepares the task script in the command window: determines the parameters for launching a task, configures the working environment necessary for the application to work, and determines the task launch sequence. After that, the task is placed in the corresponding queue.

Job configuration supports a large number of different parameters like the number of cores or the amount of GBs RAM per process. But the most important parameters are the amount of compute resources (nodes) and the time of their allocation.

Additionally, the number of processors per subtask, the amount of memory in the node, the amount of memory in the processor, the number of subtasks in the task, the start time of the task, and others can be specified. The user has the ability to select a variety of options for a detailed description of how the task should be performed.

The Slurm forms a separate job queue for each type of SCC computing resource. While creating a task, users independently select the required resources. Due to the qualification of most users in the field of parallel programming, they prefer to use traditional CPU-based nodes. And often they use only 1–2 nodes for their task. Thus, different clusters within the SCC are loaded differently. An example of the loading of computing clusters of the SCC "Polytechnic" is shown in Fig. 2.

All tasks were divided into classes depending on the field of science: Astrophysics, Bioinformatics, Biophysics, Energy, Geophysics, Informatics, Mechanical Engineering, Mechanics, Physics and Radiophysics. For clarity, in Fig. 2, each of the classes is indicated by its own color:

- Astrophysics is turquoise;
- Bioinformatics is red;
- Biophysics is pink;
- Energy is green;
- Geophysics is yellow;
- Informatics is black;
- Mechanical Engineering is blue;
- Mechanics is gray;
- Physics is lime;
- Radiophysics is orange;
- Uncertain Tasks are purple.

## 2. Approaches to Improving the Efficiency of the SCC

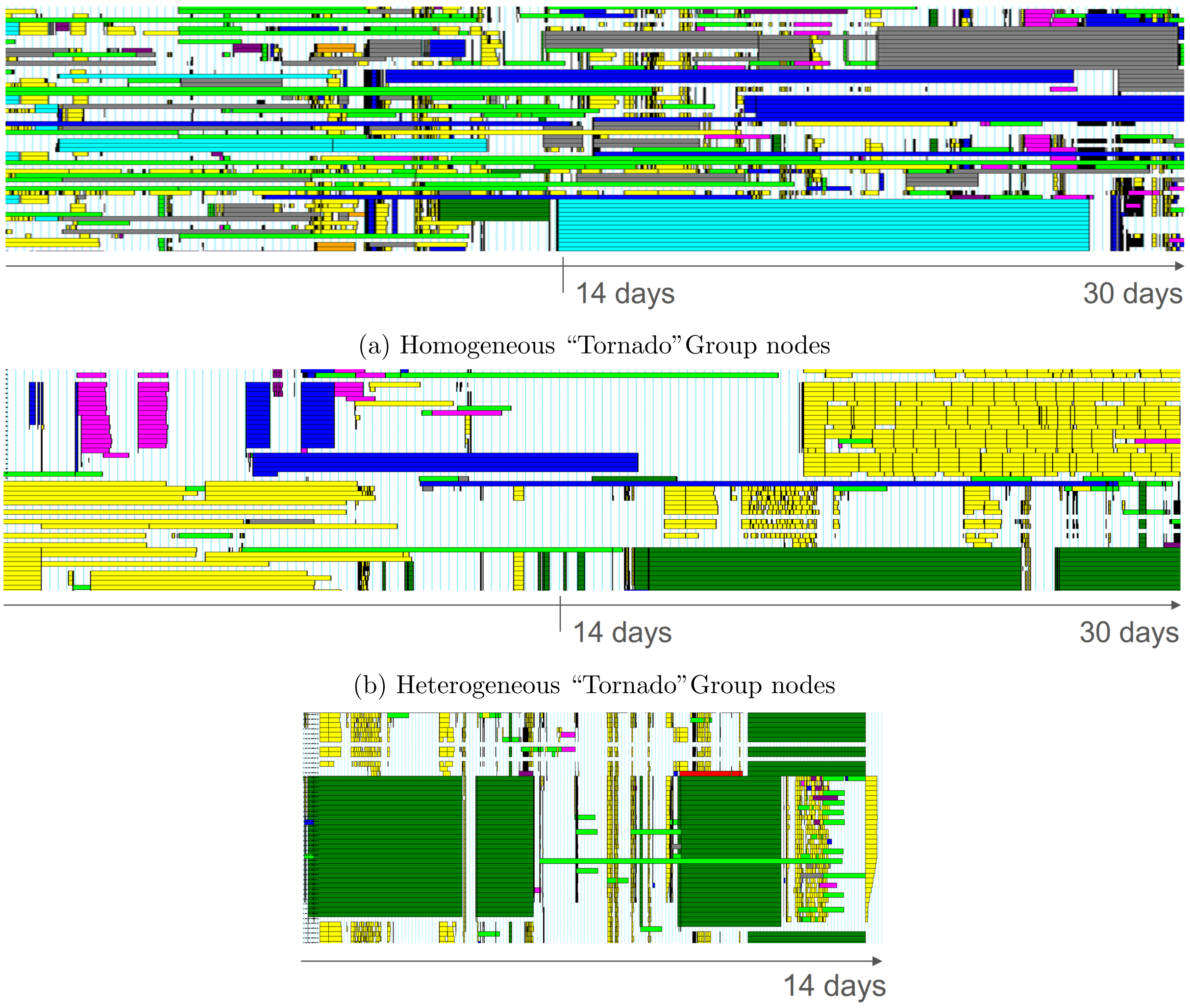
One of the promising directions for increasing the efficiency of using supercomputer resources is the use of various task parameters to adapt the process of its solution.

Currently, SPbPU considers two main areas of adaptation of the work of the SCC:

1. The use of intelligent technologies and machine learning methods for a preliminary assessment of the effectiveness of solving a new problem based on statistical data on solving similar problems in the past.
2. Development of methods for adapting the SCC hardware to the requirements of a specific task by using reconfigurable computers.

### 2.1. Application of Machine Learning Methods to Optimize the Parameters of Tasks Received by the SCC

In this work using a machine learning approach to improve the real performance of SCC is proposed. The hybrid supercomputer cluster should act as an active component of the machine learning ecosystem (Fig. 3). In this mode, the supercomputer generates a multiset of calculation data, as well as information on how the application tasks affect the cluster resources. Although all data are in principle available to users, their interpretation requires high skill and, therefore, is much more difficult in practice. Obviously, most of this data can be used for machine learning



(a) Homogeneous "Tornado" Group nodes

(b) Heterogeneous "Tornado" Group nodes

(c) Homogeneous "Cascade" Group nodes

**Figure 2.** Uneven loading of cluster nodes of the supercomputer center "Polytechnic"

of intelligent blocks cluster scheduler, including for constructing a user qualification model that characterizes the description confidence level and script parameters for the executable task.

As a result, the intelligent block of the Slurm scheduler can generate estimates of the efficiency of using available resources, as well as generate a statistical metric of confidence in the results obtained, including recommendations for improving the efficiency of calculations. All data generated by the intelligent block of the Slurm scheduler, along with the direct results of calculations, are provided to users, and are also taken into account as parameters in the trained model of his qualification profile.

The machine learning ecosystem infrastructure built in this way, which includes both the supercomputer itself and its various users, allows you to analyze the entire course of calculations performed and fixing the trajectory of each application process, organizing an effective machine learning process of the resource scheduler, including for the tasks of new users.

In this case, the training sample includes both "successful" and "unsuccessful" sets of completing tasks, that carry out important information characterizing the so-called "survivor's error". Generating an explanation of why the application task "survived" or why it was not completed within the time specified by the scheduler will speed up the user's understanding of the

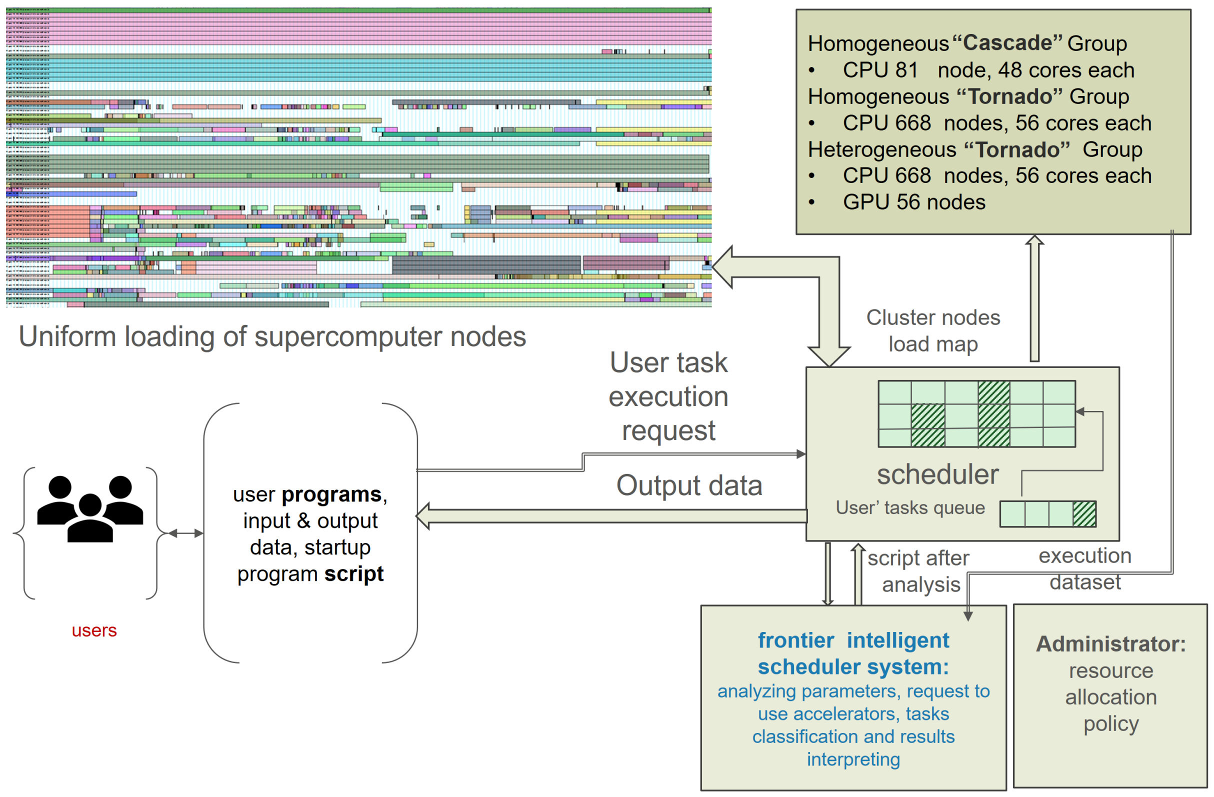

Uniform loading of supercomputer nodes
Homogeneous "Cascade" Group
• CPU 81 node, 48 cores each
Homogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
Heterogeneous "Tornado" Group
• CPU 668 nodes, 56 cores each
• GPU 56 nodes
Cluster nodes load map
User task execution request
Output data
scheduler
User' tasks queue
user programs, input & output data, startup program script
users
script after analysis
execution dataset
frontier intelligent scheduler system: analyzing parameters, request to use accelerators, tasks classification and results interpreting
Administrator: resource allocation policy

**Figure 3.** Structure and information flows of the supercomputer center "Polytechnic"

specifics of using supercomputer resources. At the same time, the dispatcher will be able to predictively calculate the parameters of the description of the applied task, which increase the probability of its successful completion, as well as "learn"to respond to new tasks, including reconfiguration of accelerators used to solve special applied tasks.

## 2.2. Multi-agent Scheduler of Reconfigurable Computational Resources

To expand the capabilities of the SCC in terms of solving specialized tasks, a cluster of Tertius-2 nodes containing FPGA appeared as part of our hybrid SCC in 2022. The work on the integration of the reconfigurable nodes (reconfigurable computation units – RCU) into the computational field of the SCC is in progress. The following scheme for integrating RCU into the structure of the SCC "Polytechnic"is proposed in Fig. 4. The advantage of the proposed scheme is the presence of regular mechanisms for integrating the RCU into the computational field of the SCC "Polytechnic", as well as the implementation of the possibility of direct control of RCU by the Multi-agent scheduler after the initialization procedure.

According to research works [11, 12] the multi-agent scheduler (MAS) is an effective decentralized method of managing a distributed computing cluster, including a heterogeneous one.

In general terms, the process of solving problems with the use of RCU consists of the following steps:

1. Upon request from the MAS RCU to Slurm, the latter allocates RCU resources to full control of the MAS by launching a software agent on each allocated RCU.
2. Among the tasks received from the user, the AI module for reconfigurable system (AIM) selects those that can be solved in the RCU using the RCU firmware that exist in the

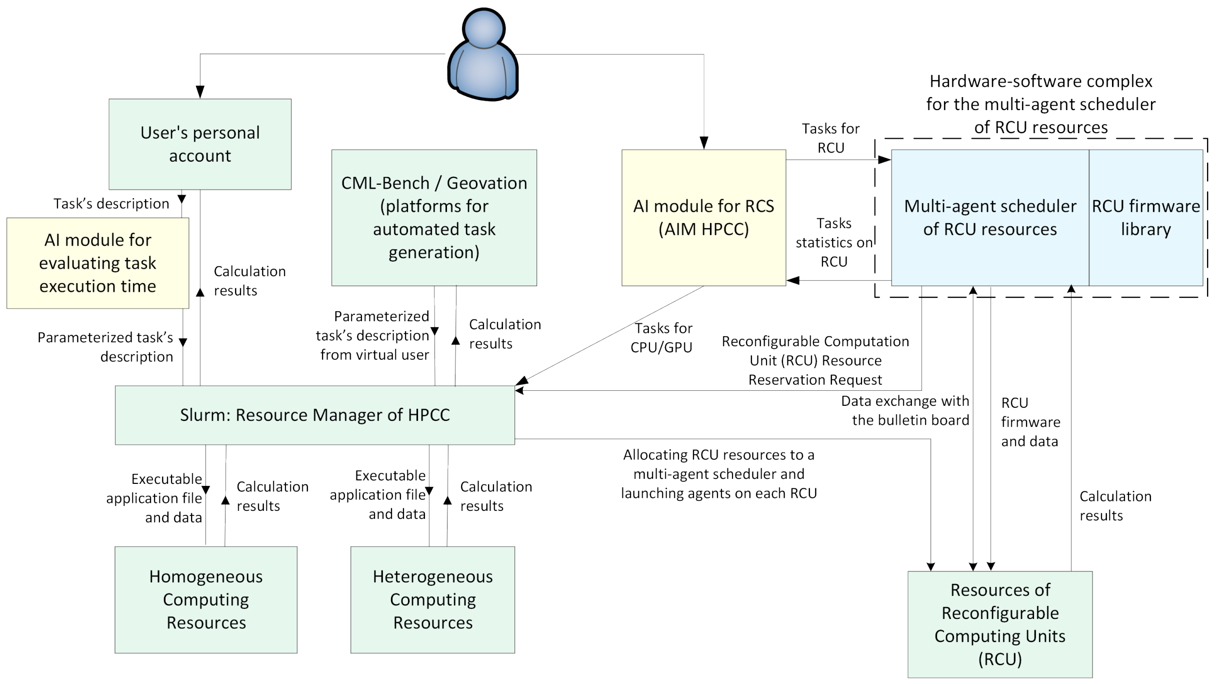

User's personal account
Task's description
AI module for evaluating task execution time
Calculation results
Parameterized task's description
CML-Bench / Geovation (platforms for automated task generation)
Parameterized task's description from virtual user
Calculation results
AI module for RCS (AIM HPCC)
Tasks for RCU
Tasks statistics on RCU
Hardware-software complex for the multi-agent scheduler of RCU resources
Multi-agent scheduler of RCU resources
RCU firmware library
Tasks for CPU/GPU
Reconfigurable Computation Unit (RCU) Resource Reservation Request
Data exchange with the bulletin board
RCU firmware and data
Slurm: Resource Manager of HPCC
Allocating RCU resources to a multi-agent scheduler and launching agents on each RCU
Executable application file and data
Calculation results
Executable application file and data
Calculation results
Calculation results
Homogeneous Computing Resources
Heterogeneous Computing Resources
Resources of Reconfigurable Computing Units (RCU)

**Figure 4.** Proposal for the integration of reconfigurable nodes under the control of a multi-agent scheduler into the SCC "Polytechnic"

firmware library. A task descriptor is formed as a set of parameters characterizing the selected task.

3. AIM, if necessary, decomposes the input data of the task into blocks that can be processed by the RCU, and for each of the blocks, the preferred solution method is selected from the existing RCU configuration modules in the library.
4. AIM transmits the generated task descriptions with a descriptor to any of the bulletin board.
5. Each software agent on the RCU independently polls all bulletin boards to find the most suitable user tasks. For each of the tasks on the bulletin board, using the information generated by the AIM, the agent determines whether it can be performed.
6. If the agent can compute the task, it, using information from the AIM, determines which RCU configuration module from those available in the library it should use for this.
7. If the agent decides to solve a problem, if necessary, it downloads the recommended RCU configuration module from the library and loads it into the RCU, executes the task in the RCU computational field, and sends the results to the required address.
8. The statistics of the task execution time on the RCU are stored for the subsequent use of additional training of the AIM.

In the course of future work, it is planned to consider the possibility of IM integration directly on each MAS software agent.

As part of the work in 2023, the library will contain at least two RCU firmware designed for solving matrices up to 4000 by 4000 using the LU decomposition method and solving diagonal matrices with 3, 5, and 7 diagonals using the Jacobi method. In the future, the list of RCU firmware in the library can be expanded.

Upon request from MAS to Slurm, the latter allocates RCU resources for full management by launching a certain software agent on each allocated RCU (it is a task for Slurm).

With the use of AIM, among the tasks arriving at the SCC, there are computationally time-consuming tasks that can be solved in the RCU using the configuration modules existing in the library. The data of tasks, if necessary, are divided in the AIM of the SCCs into parts corresponding to the capabilities of the RCU. As a result, a task is formed with a descriptor (set of parameters) for solving on the Tertius-2 RCU, and all this is transferred to any of the MAS bulletin boards to ensure the possibility of a parallel solution in the RCS.

A software agent (hereinafter PA) is software for managing and adapting resources of a heterogeneous computing cluster for solving problems of a dynamic search for a load for an agent-controlled computing resource and adaptive reconfiguration of hardware components of the RCU, constantly running on the universal processor of the RCU and, together with other MAS agents, forming a single computing – communication space.

The software agent can receive data about its parameters from the RCU. In addition, the SCC already has software that allows you to monitor the parameters of all RCUs.

Each of the PA, in case of readiness of its RTM for work and the absence of tasks currently performed on its RTM, independently interrogates all DOs to find the most suitable user tasks.

For each of the tasks on the DO, using the information generated by the RCC IM and attached to the task, the agent determines whether it can complete it. If it can fulfill the task, it, using information from the IM SKTS, determines which RCU configuration modules from those available in the library must be used. If the agent decides to solve a problem, if necessary, it downloads the configuration module from the library and loads it into the RCU, executes the task in the RCU computational field, and sends the results to the required address.

Depending on the mode of operation of the RCS, in the event of a long idle time or at the command of the operator, the agent can complete its work and return control to the RBC Slurm.

The first step in increasing the efficiency of a supercomputer is to minimize the average time jobs are spending in the queue. For making a denser queue, Slurm needs a reliable information about the task execution time. As studies have shown, the results of which are given below, users significantly overestimate the task execution time and the Slurm is forced to look for a larger slot in queue to put the task in it. This situation negatively affects the length and density of the queue.

Machine learning models are proposed to be used for more accurate estimation of the task execution time. The input of such models is the vector of task's parameters $\mathbf{x}_i = (x_{i1}, ..., x_{im})$, and the output is the expected execution time of this task. The values of the individual parameters of the task $i$ in vector $\mathbf{x}_i$ are taken from the data of the Slurm processes: scontrol, sacct, and sbatch. For example, $x_{i1}$ is the ID of the user running the task, and $x_{i2}$ is the ID of the user's group, followed by the requested number of computing nodes, and so on, including meta-parameters of the executable file, such as included libraries. A description of the machine learning methods used in the work is given below.

## 3. Some Elements of Survival Analysis and Preliminaries

### 3.1. Basic Concepts of Survival Analysis

We represent a dataset $D$ of tasks in the framework of survival analysis as a set of triplets of the form $(\mathbf{x}_i, \delta_i, T_i)$, where $\mathbf{x}_i = (x_{i1}, ..., x_{im})$ is the feature vector which contains all available information about the $i$-th task represented by $m$ features; $T_i$ is the $i$-th task completion time.

In contrast to the standard regression analysis, there is the additional component $\delta_i$ of the dataset which is the indicator function so that $\delta_i = 1$ if we observe a successful task completion, and $\delta_i = 0$ if the $i$-th task has not been successfully completed. In the first case ($\delta_i = 1$), time $T_i$ corresponds to the time between the baseline time and the time of the successful task completion. This case corresponds to the uncensored observation. In the second case ($\delta_i = 0$), $T_i$ is the observation time, i.e., the time moment when the task is terminated due to several reasons, and we have the censored observation. The aim of survival analysis is to predict the completion time of a new task characterized by the feature vector $\mathbf{x}$ by using the training dataset consisting of $n$ examples $(\mathbf{x}_i, \delta_i, T_i)$, $i = 1, ..., n$.

Important concepts in survival analysis are the survival and hazard functions. The survival function, denoted as $S(t|\mathbf{x})$, is the probability that the task $\mathbf{x}$ is not completed up to the time $t$, that is $S(t|\mathbf{x}) = \Pr\{T > t|\mathbf{x}\}$. The hazard function or the hazard rate $h(t|\mathbf{x})$ is the rate of the task completion at time $t$ given that no tasks completed before time $t$. It can be written as

$$h(t|\mathbf{x}) = \lim_{\Delta t \to 0} \frac{\Pr\{t \leq T \leq t + \Delta t | T \geq t|\mathbf{x}\}}{\Delta t} = \frac{f(t|\mathbf{x})}{S(t|\mathbf{x})}, \tag{1}$$

where $f(t|\mathbf{x})$ is the density function of the task completion.

The hazard function can also be expressed though the survival function as follows:

$$h(t|\mathbf{x}) = -\frac{d}{dt} \ln S(t|\mathbf{x}). \tag{2}$$

Hence, we can express the survival function through the cumulative hazard function $H(t)$ as

$$S(t|\mathbf{x}) = \exp\left(-\int_0^t h(z|\mathbf{x})\mathrm{d}z\right) = \exp\left(-H(t|\mathbf{x})\right). \tag{3}$$

It can be seen from the above that the functions depend on the vector $\mathbf{x}$.

There are several well-known models used in survival analysis. First, we should mention the non-parametric Kaplan–Meier model for estimating the survival function. However, the Kaplan–Meier estimator gives the average view of objects (tasks) and does not take into account the task features $\mathbf{x}$. Therefore, we cannot evaluate how the task features influence the survival probabilities. The first model which allows us to estimate the survival or hazard functions depending on the vector $\mathbf{x}$ is the Cox proportional hazards model [4]. According to the model, the hazard function at time $t$ given the feature vector $\mathbf{x}$ is defined as

$$h(t|\mathbf{x}) = h_0(t) \exp\left(\mathbf{x}\mathbf{b}^{\mathrm{T}}\right), \tag{4}$$

where $h_0(t)$ is an arbitrary baseline hazard function; $\mathbf{b} = (b_1, ..., b_m)$ is an unknown vector of regression coefficients or parameters.

The survival function $S(t|\mathbf{x})$ is computed as

$$S(t|\mathbf{x}) = \left(S_0(t)\right)^{\exp\left(\mathbf{x}\mathbf{b}^{\mathrm{T}}\right)}. \tag{5}$$

Here $S_0(t)$ is the baseline survival function. It is important to point out that $S_0(t)$ as well as $h_0(t)$ do not depend on $\mathbf{x}$ and are estimated by using the Kaplan–Meier estimator.

The main peculiarity of the Cox model is the linear combination of features. On the one hand, it simplifies the model and allows us to use it in the interpretation of the model predictions. On the other hand, it restricts the Cox model use because real datasets usually have a more complex structure. Various modifications have been proposed to generalize the Cox model by replacing the linear relationship with some non-linear functions, for example, with neural networks [5, 20], the support vector machine [14, 22].

### 3.2. Random Survival Forests

Despite the availability of many machine learning models for survival analysis, they require a large amount of training data to obtain reasonable estimations. One of the efficient models dealing with small and heterogeneous data is the random survival forests [10]. A general algorithm of constructing RSFs can be represented as follows:

1. $Q$ bootstrap samples of size $N$ are selected from the original data, where $Q$ is a hyperparameter defining the number of survival trees in the random survival forests.
2. A survival tree is constructed by using a single bootstrap sample so that each node of the tree is split using the candidate feature that maximizes survival difference between daughter nodes. The depth of the trees is also a hyperparameter.
3. The survival function or the cumulative hazard function at each leaf of a tree is calculated by using the Kaplan–Meier estimator or the Nelson–Aalen estimator.
4. The ensemble cumulative hazard function or the ensemble survival function is obtained by averaging the cumulative hazard functions over all trees.

One of the important peculiarities of the tree construction is a splitting rule. Several splitting rules are reviewed in [27]. The most popular rule is the log-rank rule which separates nodes of the decision tree by selecting the split that yields the largest log-rank test [25]. The best split is given by the greatest difference between two daughter nodes which is given by the largest value of the log-rank test.

If we denote the cumulative hazard estimate of the $k$-th tree as $H_k(t|\mathbf{x})$, then the ensemble cumulative hazard estimate for the random survival forest consisting of $Q$ trees is determined as follows [10]:

$$H(t|\mathbf{x}) = \frac{1}{Q}\sum_{k=1}^{Q} H_k(t|\mathbf{x}). \tag{6}$$

In fact, the above expression can be regarded as the mean function over all cumulative hazard functions predicted by each tree in the random survival forest. The obtained function will be used later for computing the survival function and the expected time to the task completion. In order to verify whether the obtained survival function is the best one from the optimal hyperparameters point of view, the C-index is used.

### 3.3. C-index

An important question in survival analysis is how to compare the machine learning survival models. One of the measures for comparison is the C-index (the concordance index) proposed by Harrell et al. [6]. The C-index allows us to estimates how good the model is at ranking survival times. This is the probability that the task completion times of a pair of tasks are correctly ranking [19]. In order to formally define the C-index, we first define inadmissible pairs. A pair is not admissible if the task completion events are both right-censored or if the earliest time in the pair is censored. The C-index is calculated as the ratio of the number of pairs correctly ordered by the model to the total number of admissible pairs. If the C-index is equal to 1, then the corresponding survival model is supposed to be perfect. If the C-index is 0.5, then the model is no better than random guessing. By using the predicted survival function $S(t|\mathbf{x})$, we can write the C-index as [27]:

$$C = \frac{1}{M} \sum_{i:\delta_i=1} \sum_{j:T_i<T_j} \mathbf{1}\left[S(T_i|\mathbf{x}_i) > S(T_j|\mathbf{x}_j)\right]. \tag{7}$$

Here $M$ is the number of all comparable or admissible pairs; $\mathbf{1}[\cdot]$ is the indicator function taking the value of 1 if its argument is true, and of 0 – otherwise.

Another definition of the C-index is

$$C = \frac{\sum_i \sum_j \mathbf{1}\left[T_i < T_j\right] \cdot \mathbf{1}\left[\widehat{T}_i < \widehat{T}_j\right] \cdot \delta_i}{\sum_i \sum_j \mathbf{1}\left[T_i < T_j\right] \cdot \delta_i}, \tag{8}$$

where $\widehat{T}_i$ is the predicted survival duration; $i$ and $j$ are indices of all examples from the dataset, $i, j = 1, ..., n$.

## 4. Algorithm for the Task Completion Prediction

A goal of the proposed algorithm of training the random survival forest is to predict the probabilistic characteristics of the task completion, including the survival function and the expected task completion time. At the first stage of study when the dataset of examples is restricted, we propose to apply the random survival forest as one of the efficient models dealing with the limited number of training data. The random survival forest consists of $Q$ survival trees.

Every vector $\mathbf{x}$ consists of $m$ features which include the most important ones: UserID (the ID of a user), GroupID (the ID of the group on behalf of which the task was queued), NumNodes (the number of nodes requested or allocated for the task), NCPUs (the total number of processors allocated to the task), NumTasks (the total number of subtasks in the task), CPUs/Task (the ratio of the total number of processors to the number of subtasks), ReqB:S:C:T (the number of different hardware components requested for the task), Socks/Node (the desired ratio of the number of sockets to the number of compute nodes), NtasksPerN:B:S:C (the number of subtasks required to run on a specific number of hardware components), CoreSpec (the number of cores reserved), MinCPUsNode (the minimum ratio of the number of processors to the number of nodes), MinMemoryNode (the minimum ratio of memory in MB to the number of nodes), JobID (the task identification number), Priority (the task priority determined by the SLURM scheduler), etc. It should be noted that the completion times $T_i$ of different tasks are changed in a large interval of time. The distribution of tasks in accordance with their completion times is shown in Fig. 5. It can be seen from this distribution that the number of "long" tasks is rather small in comparison with tasks completed in a short time. This causes a difficulty for training the random survival forest because survival trees are mainly trained on the "short" task and do not take into account the "long" tasks. In order to overcome this problem, we propose to cluster all training examples into $K$ groups which are separated by using the completion time $T$ as well as the feature vector $\mathbf{x}$. However, the completion time is more important in comparison with the feature vector because it determines the presented distribution of tasks. At the same time, the vector $\mathbf{x}$ should be also used. Therefore, we propose to introduce weights $w_0$ and $w_1$ of the completion time as well as the feature vector, respectively, in the clustering procedure so that $w_0 + w_1 = 1$, $w_0 > w_1$. The weighted K-means clustering procedure is used, where the distance

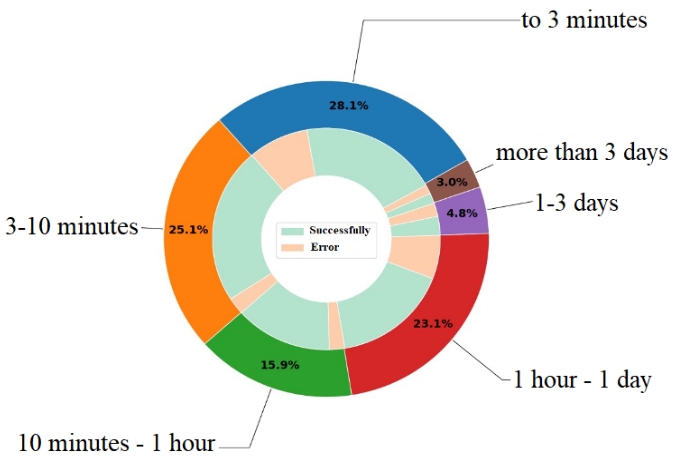


**Figure 5.** Distribution of tasks in accordance with their completion times

between the centroid $\mathbf{c}$ of points from a cluster $C_k$ and the vector $\mathbf{x}$ are computed as follows:

$$dist(\mathbf{c}, \mathbf{x} \cup T) = \sqrt{w_0 \left(T - c^{(T)}\right)^2 + w_1 \left\|\mathbf{x} - \mathbf{c}^{(f)}\right\|^2}, \tag{9}$$

where $c^{(T)}$ and $\mathbf{c}^{(f)}$ are two parts of the centroid vector corresponding to the completion time and the feature vector, respectively, so that $c^{(T)} \cup \mathbf{c}^{(f)} = \mathbf{c}$.

The cluster procedure divides the dataset $D$ into $K$ clusters $D_1, ..., D_K$ such that $D_1 \cup ... \cup D_K = D$. Now we can train $K$ random survival forests on $K$ obtained datasets (clusters). In sum, we have $K$ models for predicting the survival function. The next question is how to use these models to obtain prediction for a new task $\mathbf{x}$. We do not know the cluster for $\mathbf{x}$ and cannot determine it because we do not know the completion time $T$ for the task. Therefore, we propose the following procedure for solving this problem. The new task is fed to $K$ random survival forest, and we obtain $K$ survival functions $S_1(t|\mathbf{x}), ..., S_K(t|\mathbf{x})$. It is obvious that only one of the survival functions is correct. In order to select the unique result, we propose the following approach. By having survival functions, we compute $K$ expected times to the task completion $E_1(\mathbf{x}), ..., E_K(\mathbf{x})$, where $E_k(\mathbf{x}) = \int_0^\infty S_k(t|\mathbf{x})\mathrm{d}t$. Since the number of observations is finite, say $n$, and it is assumed that all times to the task completion are different, then this integral is reduced to the sum:

$$E_k(\mathbf{x}) = \sum_{i=1}^{n} S_k(t_i|\mathbf{x})(t_i - t_{i-1}),\ t_0 = 0. \tag{10}$$

Assuming that the time to the task completion for the new task $\mathbf{x}$ is $E_k(\mathbf{x})$, we obtain vectors $\mathbf{x} \cup E_k(\mathbf{x})$, $i = 1, ..., K$. A simple rule for selecting the cluster for $\mathbf{x}$ is to find the smallest distance between the obtained vector and the centroid, i.e., we select the $k$-th cluster and the $k$-th survival function $S_k(t|\mathbf{x})$ if there holds

$$k = \arg \min_{j=1,...,K} dist(\mathbf{c}_j, \mathbf{x} \cup E_j(\mathbf{x})). \tag{11}$$

In order to find the optimal or suboptimal hyperparameters, we use the C-index given in (7) or (8), which can be regarded as a measure of the model quality.

Finally, we can write a scheme of the following algorithm for training and testing the survival model and for computing the survival functions of a new task.

1. Initial data: dataset $D = \{(\mathbf{x}_i, \delta_i, T_i), i = 1, ..., n\}$. Hyperparameters: the number of random survival trees $Q$ in the forest, the number of examples $N$ from the dataset for constructing random trees; the number of clusters $K$, weights $w_0$ and $w_1$.
   **Training part:**
2. Divide the dataset $D$ into two subsets: training $D_{train}$ and testing $D_{test}$.
3. Divide all examples from the dataset $D_{train}$ into $K$ clusters $D_1, ..., D_K$ by using the K-means algorithm based on the distance metric given in (9) with weights $w_0$ and $w_1$.
4. Train $K$ random survival forests consisting of $Q$ trees on datasets $D_1, ..., D_K$ by randomly selecting $N$ examples for every tree, respectively.
   **Testing part:**
5. Take examples $\mathbf{x}$ from the testing subset $D_{test}$.
6. Compute the cumulative hazard function $H_i^{(k)}(t|\mathbf{x})$ for the $i$-th tree from the $k$-th random forest, $i = 1, ..., Q$, $k = 1, ..., K$.
7. Compute the ensemble cumulative hazard function $H^{(k)}(t|\mathbf{x})$ for the $k$-th random forest by using (6) for all $k = 1, ..., K$.
8. Compute the survival function $S_k(t|\mathbf{x})$ from $H^{(k)}(t|\mathbf{x})$ by using (3) for all $k = 1, ..., K$.
9. Compute expected times $E_k(\mathbf{x})$ to the task completion by using (10) for all $k = 1, ..., K$.
10. Compute distances $dist(\mathbf{c}_j, \mathbf{x} \cup E_j(\mathbf{x}))$, $j = 1, ..., K$, and find $k$ corresponding to the smallest distance.
11. Output $S_k(t|\mathbf{x})$.
12. Compute the C-index by using (7) or (8) and the testing dataset $D_{test}$. If the C-index does not satisfy the model requirements, then the hyperparameters are changed and go to Step 1, otherwise it is supposed that the model is successfully trained and can be used for new tasks.

## 5. Improving the Algorithm

One of the difficulties when the proposed algorithm is implemented is the restricted information about a user. Indeed, we did not use a history of tasks that had been previously performed by the user. This information may significantly improve the algorithm and enhance the quality of predictions.

Suppose that there are $R$ events of the task unsuccessful termination and $S$ events of the task successful completion when the user ran tasks. The unsuccessful terminations of the task may be due to the following reasons:

- termination of the task as a result of the intended stop when the program execution time exceeded the time requested by the user ($r_1$);
- termination of the task as a result of division by zero ($r_2$);
- termination of the task as a result of an infinite loop ($r_3$);
- termination of the task as a result of an unknown failure ($r_4$);
- termination of the task due to lack of memory ($r_5$).

Here $r_1, ..., r_5$ are numbers of unsuccessful terminations.

The successful completion of the task can also include the following events:

- the ratio of the program execution time and the time requested by the user does not exceed 20% ($s_1$);
- the ratio of the program execution time and the time requested by the user is between 20% and 40% ($s_2$);

- the ratio of the program execution time and the time requested by the user is between 40% and 60% ($s_3$);
- ...
- the ratio of the program execution time and the time requested by the user exceeds 100% ($s_6$).

Here $s_1, ..., s_6$ are numbers of different events of the task successful completion.

Let us construct two probability distributions $\pi = (\pi_1, ..., \pi_5)$ and $\varphi = (\varphi_1, ..., \varphi_6)$ defined as

$$\pi_i = r_i/R,\ i = 1, ..., 5,\ \varphi_j = s_j/S,\ j = 1, ..., 6.$$

The above implies that the user history can be represented by means of the above features in the form of the two distributions. Hence, we extend the feature vector $\mathbf{x}$ of a user by the probability distributions $\pi$ and $\varphi$. If a new user is analyzed, we can consider the uniform probability distribution to characterize the new user. Another way for taking into account the new user is to add two additional features which take values of 0 if the user is not new and has some history of the task performance, and of 1 if the user is new. It is interesting to point out that these additional features can be in interval $[0, 1]$ indicating uncertainty of the historical observation. In particular, when the number of runs by the user is small, then values of the two additional features should be close to 1, otherwise they are close to 0. The choice of the uncertainty measure is a direction for further research.

## 6. Interpretation of the Machine Learning Analysis Results

One of the important problems to optimize the computer performance is to explain why the user task is characterized by the obtained survival function or the expected time to the task completion. This problem can be referred to the direction called the eXplainable Artificial Intelligence (XAI). Methods of XAI try to answer the question which features of an example significantly influence a prediction of a machine learning model. Most methods are explained locally, that is, they explain a prediction of a single example, and assume that the analyzed machine learning model is a black box which means that we know only its input and output data. A lot of the explanation methods are based on local approximating the unknown prediction function at a point by means of the linear function of features because coefficients of the function can be regarded as quantitative impacts on the prediction.

One of the first local explanation methods is the Local Interpretable Model-agnostic Explanations (LIME) [23], which uses linear model to approximate predictions of black-box models. According to LIME, the explanation is derived from a set of synthetic examples generated randomly in the neighborhood of the explained example. Every synthetic example has a weight depending on its proximity to the explained example. However, LIME cannot be applied to survival machine learning models because the output of the models is not a point, but the function (the cumulative hazard function or the survival function).

To overcome this difficulty, another method called SurvLIME (Survival LIME) has been proposed in [16]. The main idea behind SurvLIME is to approximate the black-box survival model predictions by using the Cox model. It can be seen from (4) that the hazard function in the Cox model contains the linear function of features. This implies that coefficients of the linear function can be viewed as quantitative impacts on the predicted hazard function. In other words, SurvLIME uses the Cox model as an explainable meta-model or an approximation of the cumulative hazard function or the survival function predicted for an example by the black-box model. According to SurvLIME, a set of examples $\{\mathbf{x}_1, ..., \mathbf{x}_L\}$ around the explained point $\mathbf{x}$ are generated and then fed to the black-box survival model, which produces a set of cumulative hazard functions $\{H(t|\mathbf{x}_1), ..., H(t|\mathbf{x}_L)\}$. Simultaneously, the cumulative hazard functions $H_{\text{Cox}}(t|\mathbf{x}_k, \mathbf{b})$, $k = 1, ..., L$, of the Cox model as functions of coefficients $\mathbf{b}$ are computed for all generated examples $\{\mathbf{x}_1, ..., \mathbf{x}_L\}$ by using(4). The parameters $\mathbf{b}$ of the Cox model are calculated by minimizing the average distance between the cumulative hazard functions of the Cox model and the black-box survival model. Many numerical results with using real open datasets provided in [16] demonstrate that SurvLIME is an efficient method for explaining the survival models. Moreover, following the results obtained in [16], an open-source Python package that implements the SurvLIME algorithm has been presented in [21]. Another method for explaining the survival model predictions is called SurvNAM [24]. It is based on applying of the generalized additive model $g_1(x_1) + ... + g_m(x_m)$ instead of the simple linear model. Here $g_i$ is a univariate shape function of one variable (feature). Two ideas underlying SurvNAM are the following. First, the functions $g_i(x_i)$ are usually unknown and any assumptions about their form may lead to incorrect results. Therefore, it is proposed to implement the functions by means of simple neural networks with a single-feature input. This idea allows to implement arbitrary functions. The training weights of the networks are parameters of the explanation model. In sum, the whole model represents $m$ fully connected neural networks implementing the functions $g_i(x_i)$, which are connected by the summation operation. The second idea is that the whole explanation neural network model implementing the generalized additive model is trained again by using the extended Cox model where the linear combination of features $\mathbf{x}\mathbf{b}^{\mathrm{T}}$ is replaced with the generalized additive model $g_1(x_1) + ... + g_m(x_m)$.

Finally, the algorithm of the explanation model SurvLIME in terms of the considered task completion problem is the following.

1. Initial data: the explained example $\mathbf{x}$; the standard deviation $\sigma$ for generating the perturbed examples; the number of the generated examples.
2. Generate random examples $\mathbf{x}_1, ..., \mathbf{x}_L$ around the explained point $\mathbf{x}$ having the normal distribution with the expectation $\mathbf{x}$ and the predefined standard deviation $\sigma$.
3. Find the cumulative hazard functions $H_{k_1}(t|\mathbf{x}_1), ..., H_{k_L}(t|\mathbf{x}_L)$ as predictions of the random survival forest (the random forest). Index $k_j$ corresponds to the optimal cluster which is selected when the generated example $\mathbf{x}_j$ is fed to the black-box model (the random forest).
4. Find the vector $\mathbf{b}_{opt}$ which minimizes the function of distances:

$$\mathbf{b}_{opt} = \arg\min_{\mathbf{b}} \sum_{j=1}^{L} dist(H_{k_j}(t|\mathbf{x}_1), H_{\text{Cox}}(t|\mathbf{x}_k, \mathbf{b})).$$

Results of the interpretation by means of SurvLIME and SurvNAM allow the user to determine which features significantly influence the survival function or the expected time to the task completion, and how the feature can be changed to obtain the reasonable survival function which ensures to complete the task in a required time with a certain probability.

## 7. Analysis of Tasks Statistics and Survival Functions

A structured form of the computational process metadata was obtained. It includes the parameters for launching the task, which the user has specified when have queued the task for execution to the Slurm. The considered form of data representation can be used in the machine learning methods for task schedulers.

A JSON file structure has been developed. It consists of all 89 possible attributes for tuning the sbatch SLURM command of version 21.08. Attributes, according to the custom entity, have been structured and distributed into 5 different groups: User Information, Job Accounting, Resource Management, Job Control, and Job Interaction.

The structure of the database for storing data about the tasks launched on the SCC was also proposed. It consists of three tables scontrol, sacct and sbatch, corresponding to the sources of the collected data. Each table contains an integer field ID of the task, as well as two fields of the VARCHAR(500) type, which allows us to dynamically allocate memory for the size of the attribute loaded into it, not exceeding 500 characters. These two fields allow to specify a key-value pair for an attribute and its corresponding value.

A special program was also developed for extracting, lexical processing and sending task data to the developed database. The program has been tested on a laboratory stand by running test tasks. As a result, 60 attributes are collected in the scontrol table for each task, 107 attributes are collected in the sacct table, and in the sbatch table as many records are stored as the number of parameters specified by the user during the startup script. The user, on average, specifies no more than 8 parameters in the startup script. In total, for each task ID, the database contains about 180 parameters, while some fields may be empty if Slurm was unable to obtain information on these attributes.

A program for the collection of statistical data of the SCC was developed. The program is called by the event of allocation of resources for the task, as well as by the event of the completion of the task. Thus, at the output of the program, two corresponding databases are formed. The running time for each call is about 4–5 seconds due to waiting for requests from the SCC control computer to the remote database server, so the program runs in the background so as not to cause any delay in the scheduler. As a result of the work of the statistics collection program, more than 400,000 tasks from users were collected in 2022, and more than 200,000 tasks were collected in 2023.

To solve the problem of optimizing the structure of the queue, machine learning models have been developed and implemented to predict one of the parameters of the task: its execution time, taking into account the current load of the SCC nodes. During the analysis of the data, the following problems were identified:

- a large spread of the estimated execution time value (from several seconds up to two weeks);
- a strong imbalance of tasks across the ranges to which they belong (more than 53 percent are tasks that take less than 10 minutes);
- insufficient amount of information in the available factors on which the target value is estimated.

Various machine learning models have been implemented, including regression models, data classification and clustering models, and survival models, which also allow the use of data on tasks that were forced to complete by Slurm.

Figure 6 shows the error matrix based on user ratings, normalized by the number of tasks falling within each range. The matrix shows that more than 89 percent of all tasks with an actual duration of [0, 10] minutes indicate an approximate execution time from 1 hour to 15 days. In most cases, users significantly overestimate the time it takes to complete tasks.

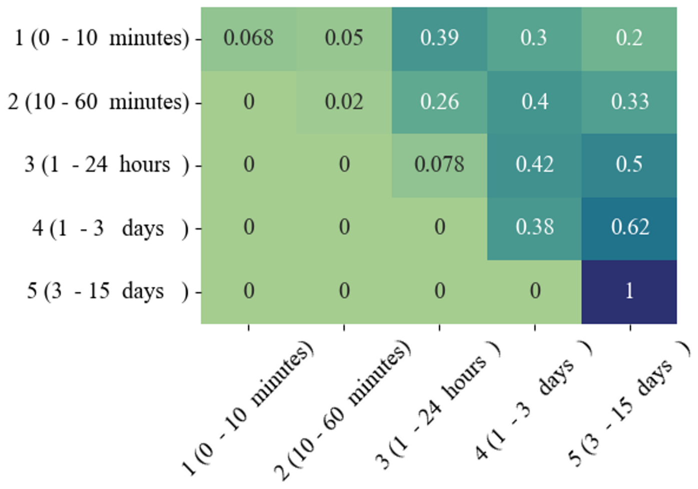


**Figure 6.** Error matrix based on user ratings, normalized by the number of tasks falling within each range

Users behavior studies were conducted to assess the impact of these data on the tasks. Users and virtual users of geovation were separated due to significant differences in their data. The following estimations were received:

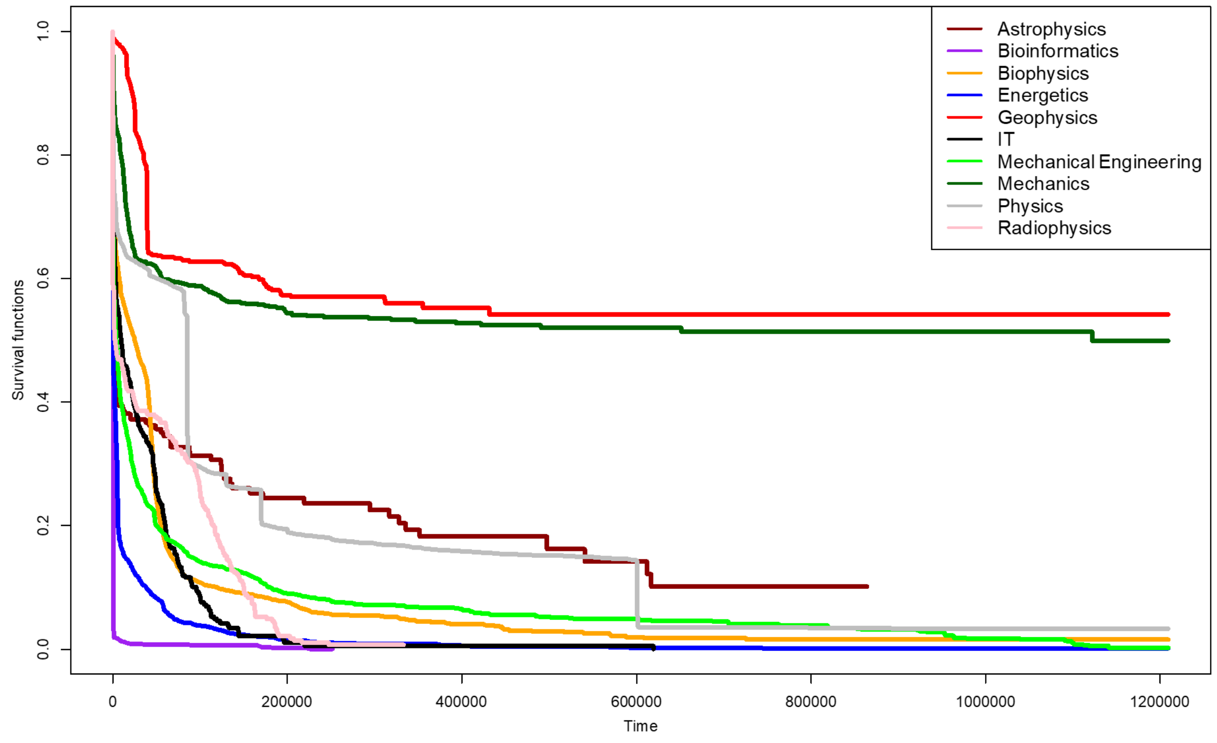
- more than 90 percent of users posted less than 100 tasks;
- more than 60 percent of virtual users of geovation posted from 30 to 60 thousand tasks and two posted more than 100 thousand tasks;
- more than 64 percent of users completed tasks in a very short time (less than 10 minutes);
- more than 92 percent of non-geovation users used just one compute node for their tasks.

The information obtained during the research showed that some of the tasks are executed exactly at the time that the user set and after that Slurm forcibly terminated them. So there is a set of tasks whose real execution time is not known, since they were forcibly terminated before they were completed. Survival models are used to work with such data. Kaplan–Meier model was used to estimate the distribution of the required time and processor time for tasks in various areas of knowledge (Fig. 7). Visual analysis of the obtained estimations of survival functions in various fields of knowledge allows to establish that tasks in the fields of geophysics and mechanics require stochastically more time to be successfully completed than tasks from other fields of knowledge, and the shortest tasks in terms of processor time usage are typical for bioinformatics.

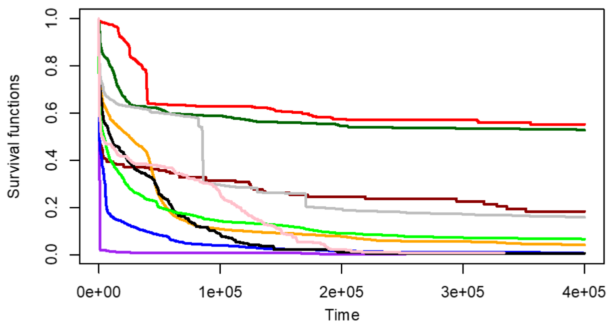
## Conclusion

Obviously, an NLP outline of the description of applied tasks can be added to the machine learning ecosystem so that the user can conduct a dialogue with the supercomputer in terms of meaningful queries in the context of applied tasks. Using the capabilities of a pre-trained transformer (Fig. 8), which generates the source code of the description of the applied task in response to the meaningful request, the user can analyze the accuracy of the formulated queries and in a recursive mode, conducting a meaningful interaction with a supercomputer, that inevitably improves his qualifications and task understanding.

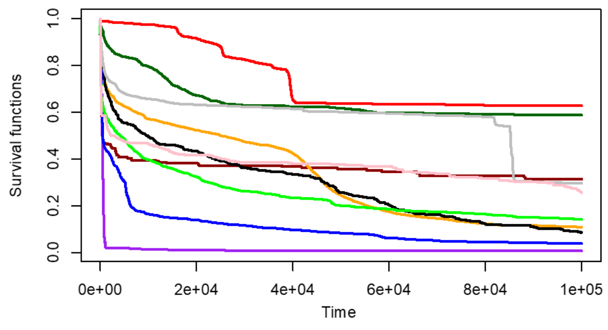
Let us point out several perspective directions for further research and development.



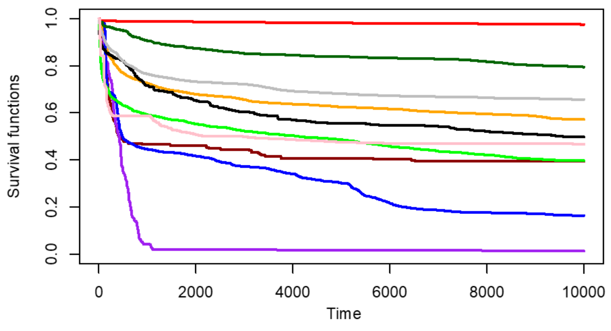
(a) $1.2 * 10^6$ seconds



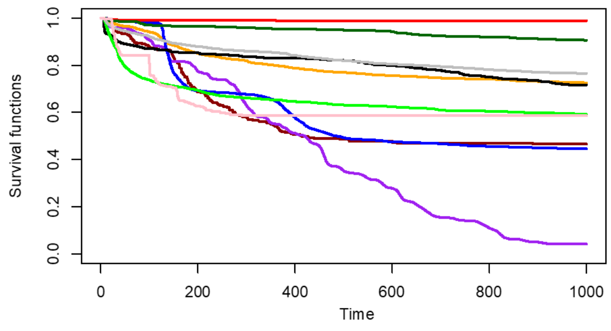
(b) $4 * 10^5$ seconds



(c) $1 * 10^5$ seconds



(d) $1 * 10^4$ seconds



(e) $1 * 10^3$ seconds

**Figure 7.** Kaplan–Meier estimations of task execution time distributions in various fields of knowledge with detailed area of task execution times not exceeding

We have proposed the specific weighted scheme of clustering, which allows us to take into account the difference between predicted times of the task completion as well as the difference between the feature vectors. However, a more interesting approach is to assign weights to each feature. Moreover, these weights should not be hyperparameters, but they have to be trained

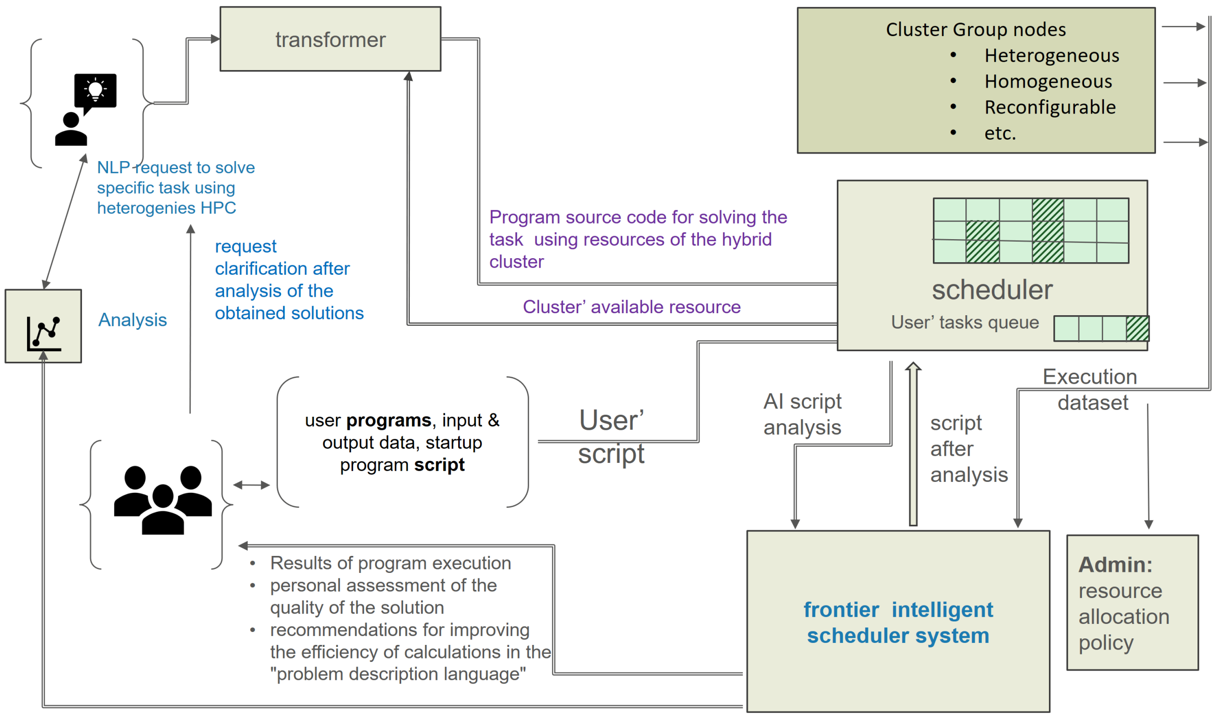

transformer
Cluster Group nodes
Heterogeneous
Homogeneous
Reconfigurable
etc.
NLP request to solve specific task using heterogenies HPC
request clarification after analysis of the obtained solutions
Program source code for solving the task using resources of the hybrid cluster
Cluster' available resource
scheduler
User' tasks queue
Analysis
user programs, input & output data, startup program script
User' script
AI script analysis
script after analysis
Execution dataset
Results of program execution
personal assessment of the quality of the solution
recommendations for improving the efficiency of calculations in the "problem description language"
frontier intelligent scheduler system
Admin: resource allocation policy

**Figure 8.** Proposal for the integration of reconfigurable nodes under the control of a multi-agent scheduler into the SCC "Polytechnic"

during the training of the whole system. This approach has two advantages. First, it allows us to make the clustering procedure to be more flexible and enhance the prediction quality. Second, it allows us to supplement the explanation procedure. The obtained weights of features show the importance of features and answer the question of which features are important from the clustering point of view. However, in order to implement the training of weights, the random survival forests have to be replaced with neural networks to train in an end-to-end manner. This requirement leads to the second perspective which is to develop a transformer to solve the survival problems and process multi-modal data. By collecting training data, we obtain the opportunity to totally or partially avoid the random forests and to construct the neural network models. However, the idea of the task description in terms of the natural language requires the development of more complex and efficient structures based on the attention mechanism and transformers. It should be noted that transformers have been proposed to solve the survival analysis problems [8, 28]. However, these transformers do not take the peculiarities of the problems which are solved when the task completion time optimization problem is solved. Another interesting idea is to combine the random survival forests and the transformer. It has been partially solved for original random forests in [15]. However, this approach cannot be directly used in survival analysis. New approaches are needed to develop an efficient multi-modal transformer-based system.

It is important to note that one of the perspective directions is to adapt the trained system to changes in the supercomputer structure, for example, to new additional computer blocks which can be supplemented in some time. In this case, we cannot directly use the trained model and need to re-train the whole prediction system. In order to avoid that, we propose to apply ideas of the heterogeneous treatment effect [1, 17] or transfer learning [18, 29]. These approaches allow us to do hard computations for training the system when the structure of the supercomputer has been changed.

"A network is a computer", a slogan that had originally been used by Sun Microsystems in the early 1980s, became a basic truism of computer science: "computers must be networked, otherwise they are... not computers". Today we propose two new extensions of former slogan, namely "A frontier ML system is HPC", and vice versa "HPC is a frontier ML system". These two slogans have not become truism yet but they clearly reflect ideas of our article and obvious computer evolution tendency – frontier high-performance computing systems become not only a driving force of global digital transformation process, but also active part of machine learning ecosystem.

## Acknowledgements

The research is partially funded by the Ministry of Science and Higher Education of the Russian Federation as part of state assignments "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence" (topic FSEG-2022-0001).

The results of the work were obtained using computational resources of the supercomputer center in Peter the Great Saint-Petersburg Polytechnic University (`https://scc.spbstu.ru`).

## References

1. Alaa, A., van der Schaar, M.: Limits of estimating heterogeneous treatment effects: Guidelines for practical algorithm design. In: Proceedings of the International Conference on Machine Learning, pp. 129–138. PMLR (2018)

2. Bou-Hamad, I., Larocque, D., Ben-Ameur, H.: A review of survival trees. Statistics Surveys 5, 44–71 (2011). `https://doi.org/10.1214/09-SS047`

3. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001). `https://doi.org/10.1023/A:1010933404324`

4. Cox, D.: Regression models and life-tables. Journal of the Royal Statistical Society, Series B (Methodological) 34(2), 187–220 (1972). `https://doi.org/10.1111/j.2517-6161.1972.tb00899/x`

5. Faraggi, D., Simon, R.: A neural network model for survival data. Statistics in Medicine 14(1), 73–82 (1995). `https://doi.org/10.1002/sim.4780140108`

6. Harrell, F., Califf, R., Pryor, D., *et al.*: Evaluating the yield of medical tests. Journal of the American Medical Association 247, 2543–2546 (1982). `https://doi.org/10.1001/jama.1982.03320430047030`

7. Hosmer, D., Lemeshow, S., May, S.: Applied Survival Analysis: Regression Modeling of Time to Event Data. John Wiley & Sons, New Jersey (2008) `https://doi.org/10.1007/s00362-010-0360-3`

8. Hu, S., Fridgeirsson, E., van Wingen, G., Welling, M.: Transformer-based deep survival analysis. In: Survival Prediction-Algorithms, Challenges and Applications, pp. 132–148. PMLR (2021)

9. Ishwaran, H., Kogalur, U.: Random survival forests for R. R News 7(2), 25–31 (2007). `https://doi.org/10.1214/08-AOAS169`

10. Ishwaran, H., Kogalur, U., Blackstone, E., Lauer, M.: Random survival forests. Annals of Applied Statistics 2, 841–860 (2008). `https://doi.org/10.1214/08-AOAS169`

11. Kalyaev, A., Kalyaev, I., Khisamutdinov, M., *et al.*: An effective algorithm for multiagent dispatching of resources in heterogeneous cloud environments. In: 5th International Conference on Informatics, Electronics and Vision (ICIEV), pp. 1140–1142. IEEE (2016). `https://doi.org/10.1109/ICIEV.2016.7760177`

12. Kalyaev, I.A., Kalyaev, A.I. Method and Algorithms for Adaptive Multiagent Resource Scheduling in Heterogeneous Distributed Computing Environments. Autom Remote Control 83, 1228–1245 (2022). `https://doi.org/10.1134/S0005117922080069`

13. Katzman, J., Shaham, U., Cloninger, A., *et al.*: Deepsurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network. BMC Medical Research Methodology 18(24), 1–12 (2018). `https://doi.org/10.1186/s12874-018-0482-1`

14. Khan, F., Zubek, V.: Support vector regression for censored data (SVRc): a novel tool for survival analysis. In: 2008 Eighth IEEE International Conference on Data Mining. pp. 863–868. IEEE (2008). `https://doi.org/10.1109/ICDM.2008.50`

15. Konstantinov, A., Utkin, L., Lukashin, A., Muliukha, V.: Neural attention forests: Transformer-based forest improvement (Apr 2023), arXiv:2304.05980. `https://doi.org/10.48550/arXiv.2304.05980`

16. Kovalev, M., Utkin, L., Kasimov, E.: SurvLIME: A method for explaining machine learning survival models. Knowledge-Based Systems 203, 106164 (2020). `https://doi.org/10.1016/j.knosys.2020.106164`

17. Kunzel, S., Stadie, B., Vemuri, N., *et al.*: Transfer learning for estimating causal effects using neural networks (Aug 2018), arXiv:1808.07804. `https://doi.org/10.48550/arXiv.1808.07804`

18. Lu, J., Behbood, V., Hao, P., *et al.*: Transfer learning using computational intelligence: A survey. Knowledge-Based Systems 80, 14–23 (2015). `https://doi.org/10.1016/j.knosys.2015.01.010`

19. May, M., Royston, P., Egger, M., *et al.*: Development and validation of a prognostic model for survival time data: application to prognosis of HIV positive patients treated with antiretroviral therapy. Statistics in Medicine 23, 2375–2398 (2004). `https://doi.org/10.1002/sim.1825`

20. Nezhad, M., Sadati, N., Yang, K., Zhu, D.: A deep active survival analysis approach for precision treatment recommendations: Application of prostate cancer. Expert Systems with Applications 115, 16–26 (2019). https://doi.org/10.1016/j.eswa.2018.07.070

21. Pachon-Garcia, C., Hernandez-Perez, C., Delicado, P., Vilaplana, V.: SurvLIMEpy: A Python package implementing SurvLIME (Feb 2023), arXiv:2302.10571. https://doi.org/10.48550/arXiv.2302.10571

22. Polsterl, S., Navab, N., Katouzian, A.: An efficient training algorithm for kernel survival support vector machines (Nov 2016), arXiv:1611.07054v. https://doi.org/10.48550/arXiv.1611.07054

23. Ribeiro, M., Singh, S., Guestrin, C.: "Why should I trust You?" Explaining the predictions of any classifier In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144. ACM (2016). https://doi.org/10.1145/2939672.2939778

24. Utkin, L., Satyukov, E., Konstantinov, A.: SurvNAM: The machine learning survival model explanation. Neural Networks 147, 81–102 (2022). https://doi.org/10.1016/j.neunet.2021.12.015

25. Waititu, H., Koske, J., Onyango, N.: Analysis of balanced random survival forest using different splitting rules: Application on child mortality. International Journal of Statistics and Applications 11(2), 37–49 (2021). https://doi.org/10.5923/j.statistics.20211102.03

26. Wang, H., Zhou, L.: Random survival forest with space extensions for censored data. Artificial Intelligence in Medicine 79, 52–61 (2017). https://doi.org/10.1016/j.artmed.2017.06.005

27. Wang, P., Li, Y., Reddy, C.: Machine learning for survival analysis: A survey. ACM Computing Surveys (CSUR) 51(6), 1–36 (2019). https://doi.org/10.1145/3214306

28. Wang, Z., Sun, J.: SurvTRACE: Transformers for survival analysis with competing events. In: Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, pp. 1–9. ACM (2022). https://doi.org/10.1145/3535508.3545521

29. Weiss, K., Khoshgoftaar, T., Wang, D.: A survey of transfer learning. Journal of Big Data 3(1), 1–40 (2016). https://doi.org/10.1186/s40537-016-0043-6