EPJ Web of Conferences **337**, 01130 (2025) https://doi.org/10.1051/epjconf/202533701130
CHEP 2024

Surrogate Modeling for Scalable Evaluation of Distributed Computing Systems for HEP Applications

Larissa Schmid[1,*], *Maximilian* Horzela[2,**], *Valerii* Zhyla[1], *Manuel* Giffels[3], *Günter* Quast[3], and *Anne* Koziolek[1]

[1]KASTEL - Institute of Information Security and Dependability, Karlsruhe Institute of Technology
[2]II. Institute of Physics, Georg-August Universität Göttingen
[3]Institute for Experimental Particle Physics (ETP), Karlsruhe Institute of Technology

Abstract. The Worldwide LHC Computing Grid (WLCG) provides the robust computing infrastructure essential for the LHC experiments by integrating global computing resources into a cohesive entity. Simulations of different compute models present a feasible approach for evaluating future adaptations that are able to cope with future increased demands. However, running these simulations incurs a trade-off between accuracy and scalability. For example, while the simulator DCSim can provide accurate results, it falls short on scaling with the size of the simulated platform. Using Generative Machine Learning as a surrogate presents a candidate for overcoming this challenge.
In this work, we evaluate the usage of three different Machine Learning models for the simulation of distributed computing systems and assess their ability to generalize to unseen situations. We show that those models can predict central observables derived from execution traces of compute jobs with approximate accuracy but with orders of magnitude faster execution times. Furthermore, we identify potentials for improving the predictions towards better accuracy and generalizability.

1 Introduction

High energy physics at the LHC relies on the global-scale federated computing infrastructure provided by the Worldwide LHC Computing Grid (WLCG) [1] for processing and storing the sheer amounts of scientific data gathered by the LHC experiments. To achieve the high-throughput demands with limited financial resources, good performance of the computing system has to be constantly ensured. With increasing demands expected from future LHC operations and technical and strategic changes in the federation, the best design for the WLCG infrastructure still needs to be identified.

Due to the size and complexity of such infrastructures, it is not feasible to build alternative infrastructures for mere testing. In addition, due to constant uptime requirements, they cannot be reserved for test and evaluation purposes. Instead, accurate simulation of workflow executions on such infrastructures and subsequent analysis of the simulated results can be performed without disturbing the operation of the real reference infrastructure, and a variety of alternative designs can be tested with just a fraction of the original investment.

*e-mail: larissa.schmid@kit.edu
**e-mail: maximilian.horzela@cern.ch

However, there is a general trade-off between accuracy and the computational complexity, and therefore execution time, that comes with designing practical simulation models as simulations with higher accuracy generally implement more fine-granular models with greater detail. This typically leads to superlinear scaling in the execution time of the respective simulation tools with respect to the size of the simulated infrastructure. As a result, it quickly becomes unfeasible to simulate workflow executions on large infrastructures.

To circumnavigate the unfavorable scaling of such simulation tools, ML surrogate models, trained on verifiable accurate simulator outputs or data from real-world systems, that predict observables from job execution traces in constant time present a natural solution approach.

2 Data Generator DCSim

To simulate the execution of HEP workflows on parallel and distributed computing (PDC) infrastructure, we utilize the DCSim tool [2, 3] as a reference simulator[1]. It is implemented using the SimGrid [4] and WRENCH [5] simulation frameworks. SimGrid and WRENCH have been chosen since they are general-purpose, enable more accurate simulation of PDC systems than other versatile frameworks [6] while keeping the level of computational complexity at manageable levels, have been carefully validated [7–11], and are utilized by a large and active community.

DCSim takes three main essential user inputs indicated as "configurations" in Figure 1. First, a platform description following the SimGrid standard defines the network of computing resources with all its (technical) characteristics, for instance, the network of links and interconnected computers, their routing, as well as their respective network bandwidths, latencies, numbers of CPU cores, CPU speeds, disk bandwidths, storage volumes, and other parameters. Second, datasets and replicas present at the simulation start, their composition of files, sizes, and locations are specified in a dedicated input file. Third, the workloads to be executed on the specified platform while possibly processing files from datasets are defined in another dedicated steering file. The workloads are characterized by their submission time, the job collection they are composed of, their respective resource demands, and the amount of computational work that must be executed for their completion. Additionally, parameters can be given to DCSim to steer individual components of the simulation models integrated into DCSim or to adjust their calibration. In this work, we use the default calibration derived for [2]. Using these inputs, DCSim simulates the scheduling of the jobs, their execution and data processing, and returns a list of observables for each simulated job. Per default, observables are the start and end times of the jobs, the total times spent with I/O and compute operations, and the processed amounts of data.

It has been shown that after calibration, DCSim can achieve valid descriptions of real-world computing workflow executions and systems [2, 3]. However, the execution time of DCSim scales superlinearly with the size of the simulated computing system, which limits its feasibility and necessitates workarounds in studies concerning computing systems at a global scale, like the WLCG. We therefore utilize DCSim as a baseline for testing ML surrogates in simulation.

3 Machine Learning Approach

Figure 1 shows an overview of our general approach. First, we conduct simulations using DCSim and preprocess the resulting data (Section 3.1). We then train our model using the

[1]Note the exact version of the simulator published here: https://doi.org/10.5281/ZENODO.8300961

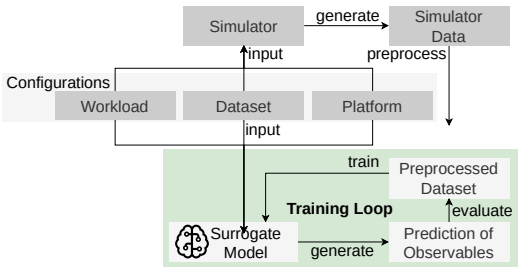


Figure 1: Approach overview.

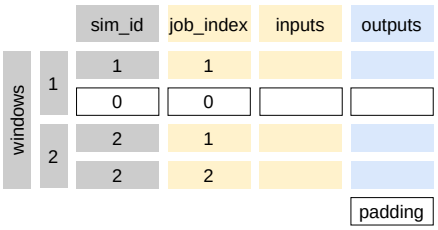


Figure 2: Layout of training dataset.

preprocessed simulator data (Section 3.2). The model receives the configuration parameters of the simulation, i.e., the execution platform, characterization of datasets used, and workload configurations, as input and predicts the observables of each job in the workload. In total, we train three different model architectures (Section 3.3).

3.1 Data Preprocessing

To create a sequential dataset that can be used to train our model, we need to preprocess the DCSim output data. First, we standardize the data to ensure that all features contribute equally to the predictions of the models, preventing features on larger scales from dominating the learning process. For each feature, we separately subtract the mean and divide the result by the standard deviation, thus adjusting the mean to zero and the standard deviation to one. We apply an inverse scaling transformation on the models' outputs to retrieve the output data at their original scale.

Next, since the number of jobs varies in each simulation and some models require fixed-length input sequences, we divide the model input vectors into windows of fixed size. If the last window of a simulation contains fewer jobs than the fixed size, we pad it with zeros to maintain a consistent window size. To be able to reconstruct the original data sequence as predicted by the simulations, we add the `simulation_ID` (not used by the model) and `job_index` (used for model training) as additional features to the entries in each input window. We provide an example illustration of a windowed dataset in Figure 2.

3.2 Model Training

We split the data into training and evaluation datasets to train the model, using a 70:30 ratio per simulation of the same length. We employ sequence-to-sequence prediction, predicting exactly one output per input row. The mean squared error (MSE) loss function is used to penalize larger errors more heavily than smaller ones, emphasizing the importance of minimizing significant discrepancies between predicted and actual values.

We adjust hyperparameters, including the hidden size, which defines the dimensionality of each hidden layer, the window size specifying the number of consecutive data points considered in each input window, and the window overlap, determining how much consecutive windows overlap. Window overlap between windows from the same simulation can be used to capture the continuity and dependencies between jobs within the same simulation, enhancing the model's ability to learn from sequential patterns. In addition, we tune the batch size, which indicates the number of input rows processed before updating the internal parameters of the model, and the number of hidden layers in the model.

We optimize these hyperparameters manually by evaluating kernel density estimation (KDE) and accuracy plots. We begin by training ten models with randomly chosen hyperparameter configurations and select the three best-performing models. For each of these models, we adjust one free hyperparameter at a time, train the model, and evaluate its performance. Once the optimal value for a specific parameter is identified, we fix it and continue with the following free parameter while retaining the best-performing model from the previous iteration. This process continues until all hyperparameters have been optimized. Finally, we compare the three resulting models and select the one that demonstrates the best overall performance. The tuning order we follow is hidden size, window size, window overlap, number of layers, and batch size.

3.3 Model Architectures

We explore three different model architectures for our surrogate model: BiGRU, BiLSTM, and Transformer models.

The BiGRU architecture consists of a linear input layer, multiple bidirectional Gated Recurrent Unit (BiGRU) layers, and a single linear output layer [12, 13]. BiGRU processes data in both forward and backward directions, making it effective in scenarios where both past and future contexts help understand the sequence. Although future jobs cannot influence already completed jobs, the training phase can benefit from data on future jobs by helping the model identify patterns or dependencies that might not be evident when considering only past and current jobs. Moreover, BiGRU can capture long-term dependencies, which is valuable for predicting simulation results, as it allows the model to account for the influence of events over extended periods. This capability can lead to more accurate predictions since jobs submitted earlier and still running can influence subsequent jobs.

The BiLSTM architecture has a structure similar to that of BiGRU but uses bidirectional Long Short-Term Memory (BiLSTM) layers instead of BiGRU layers [12, 14]. As BiGRU, BiLSTM can capture long-term dependencies and processes data in both forward and backward directions.

The transformer-based architecture we use [15] is adapted for numerical predictions. Apart from technical scalability and performance benefits, thanks to the self-attention mechanism, transformers can handle long-range dependencies and are suited better for handling local and global contexts simultaneously. In this study, we focus on encoder-only transformers, with the number of attention heads as an additional hyperparameter.

4 Evaluation of Surrogate Models

We evaluate our approach by varying the complexity of the data used for model assessment and comparing the predicted results with the ones obtained with DCSim. In the first evaluation scenario, referred to as homogeneous jobs (Section 4.1), we use jobs generated from a single workload description executed on a fixed platform with minimal complexity. This controlled setup allows us to establish a baseline for the model's performance under simple conditions. The second scenario covers heterogeneous jobs (Section 4.2), introducing increased complexity by predicting results for jobs derived from multiple workload distributions on a more complex platform. This setting provides insight into the model's capabilities to handle diverse workloads.

We show the evaluation results for the observables *compute_time* and *input_files_transfer_time* for both scenarios. The results for other observables can be found in [16].

We evaluate the models' predictions using the coefficient of determination (R-squared) and Kernel Density Estimate (KDE) plots. R-squared represents the proportion of variance in the target variable explained by the model. Its values range from zero to one, with one indicating a perfect fit. KDE plots provide a visual assessment of the distribution of the respective observable, revealing patterns or systematic bias regarding under- or overprediction that R-squared cannot capture.

4.1 Homogeneous Jobs

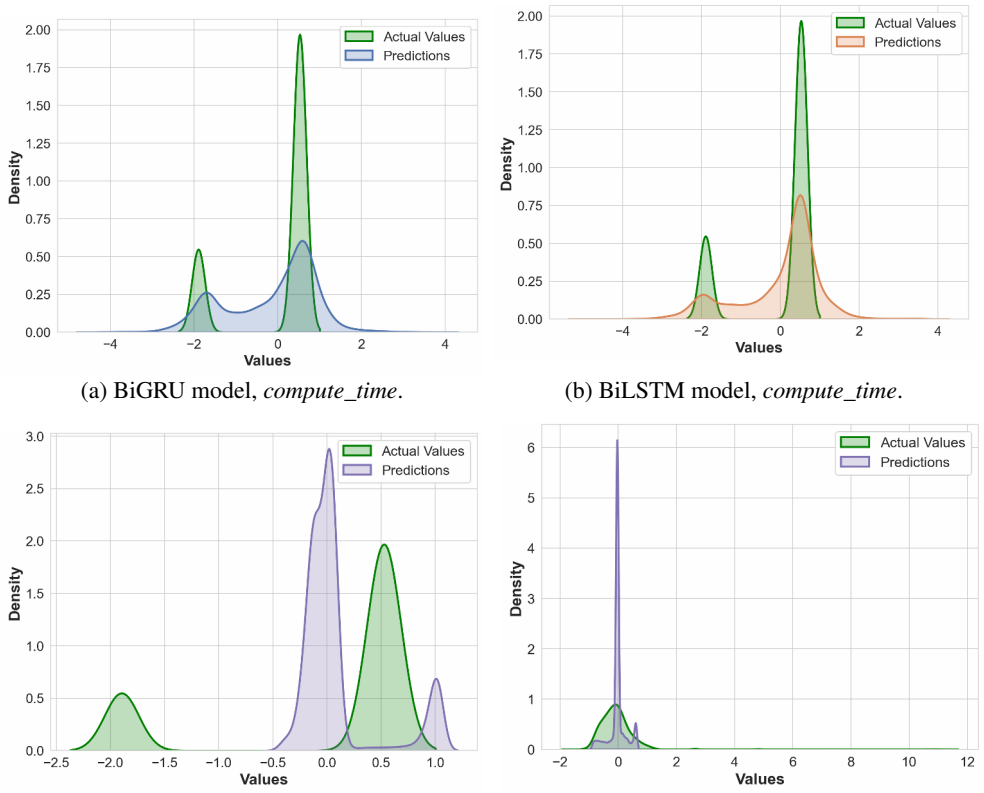
In the homogeneous jobs scenario, we use 10,000 simulations for training, organized into 10 batches of 1,000 simulations each. These simulations include 1, 10, 20, 50, 100, 250, 500, 1,000, 1,500, and 2,000 jobs per simulation. To assess extrapolation capabilities, we use an additional dataset of 10 simulations with 10,000 jobs each that are not used in training. The resource demands of all jobs remain identical throughout the training and evaluation phases. As input features, we use *index*, *flops*, *input_files_size*, and *output_files_size*. The platform configuration contains three worker nodes, two with 24 CPU cores each, one with 12 CPU cores, and one scheduler node. All nodes are connected following a star topology. The datasets processed by the jobs are stored on a separate storage server.

We show a comparison of the prediction results with the simulation for the extrapolation task in Table 1 and Figure 3. All models exhibit a negative R-squared value for *compute_time*, indicating a worse fit than a naive baseline. The KDE plots show that the target distribution exhibits two prominent spikes for the *compute_time*. While the BiGRU and BiLSTM models successfully capture the general structure of the distribution, they fail to emphasize these two spikes sufficiently. The transformer model, however, predicts mainly the mean values for *compute time*, failing to capture the actual distribution. While the R-squared value for *input_files_transfer_time* is positive, it is close to zero for all models, indicating only a slightly better fit than the naive baseline. The KDE plots show that the models capture some distribution features again but fail to emphasize the actual shape. In summary, all models failed to predict the observables beyond the first statistical moment, potentially due to the lack of input features providing additional context constraining the job execution, particularly information about the platform architecture. Although the size of the modelled infrastructure is rather minimal in this example, the surrogate ansatz already leads to improvements in the execution time of minimum two orders of magnitude on our test systems ($O(10\,\mathrm{s}) \rightarrow O(100\,\mathrm{ms})$).

4.2 Heterogeneous Jobs

Our second evaluation scenario introduces jobs from multiple distributions and a more complex platform configuration. We train the models and evaluate their extrapolation capabilities using the same strategy as for the homogeneous jobs. The workload comprises five distinct job classes, each with its own distribution for all resource demands. Each job's *submission_time* differs in this scenario and is therefore added as an input feature. The platform configuration describes an interconnected network of two data centers. The first consists of a local network of ten identical worker nodes and a storage node. The second has a single but more powerful worker node. All datasets are stored in the first data center. The first data center features 420 CPU cores, while the second has 200 CPU cores.

We present the prediction results in Table 1 and Figure 4. We omit the KDE plots for the BiGRU model since they show the same features as the BiLSTM. All models achieve an R-squared value close to one for *compute_time*, indicating near-optimal predictions. The KDE plots reveal smaller errors in matching the exact distribution but show a good fit overall despite the same lack of platform context as in the previous example. However, the R-squared



(a) BiGRU model, *compute_time*.

(b) BiLSTM model, *compute_time*.

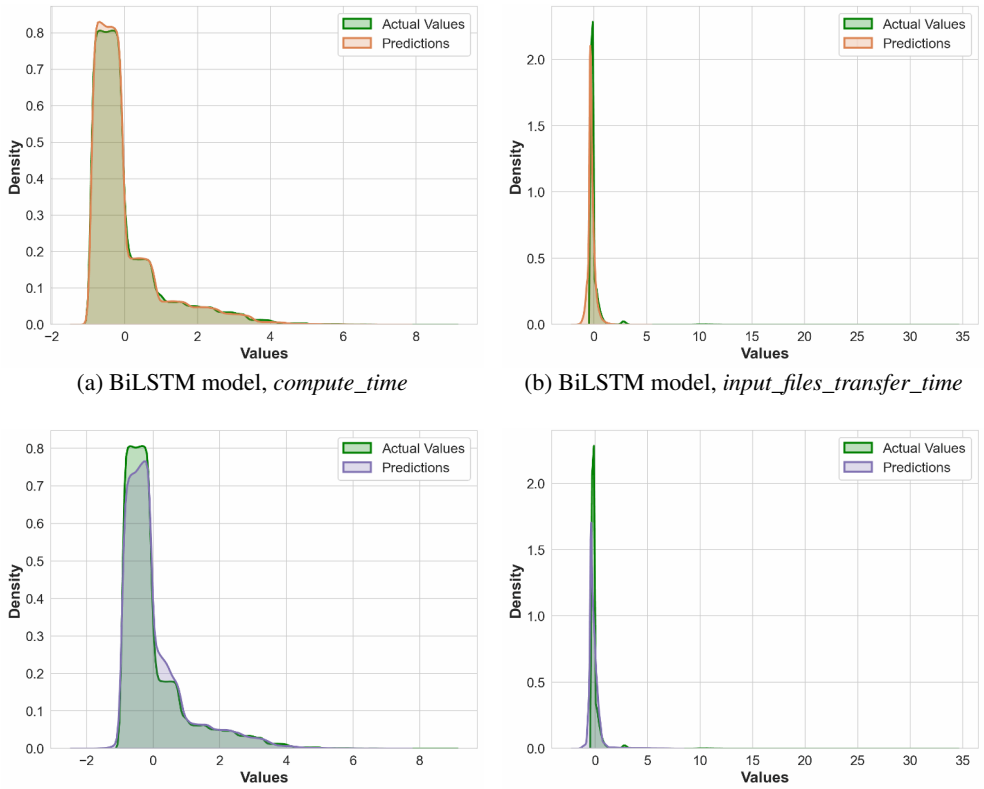
(c) Transformer model, *compute_time*.

(d) Transformer model, *input_files_transfer_time*.

Figure 3: Predictions for extrapolation of trained models in the homogenous jobs setup. The *input_files_transfer_time* is similar for the other models.

value for *input_files_transfer_time* is negative across all three models, indicating that the models cannot predict the observable effectively. Looking at the KDE plots, we can see that the models accurately predict the spikes in the distribution. However, all models fail to predict the long tail of large values, deteriorating the overall prediction quality.

Overall, all models achieve similar results, with the Transformer model achieving slightly less accurate predictions. While *compute_time* can be predicted accurately, the behavior of *input_files_transfer_time* cannot be predicted accurately. This behavior may be influenced by complex interactions within the platform, such as the existence of various routes from the node storing the dataset to the worker node that processes the job. Since information about the platform's configuration is not explicitly included in the model inputs, it cannot be utilized to make predictions. For the *compute_time*, however, the presence of multiple workloads, including jobs with different compute requirements and subsequently different execution times also depending on the worker they have been executed on, presents implicit platform information providing additional context about the platform. For the *transfer_time*, however, the effect of the platform on the job execution is more convoluted and complex than what can be captured implicitly in the training data. Nonetheless, the benefit of the surrogate in terms of execution times becomes more pronounced with a respective speed-up of multiple orders of magnitude ($O(100\,\mathrm{s}) \rightarrow O(100\,\mathrm{ms})$).



(a) BiLSTM model, *compute_time*

(b) BiLSTM model, *input_files_transfer_time*

(c) Transformer model, *compute_time*

(d) Transformer model, *input_files_transfer_time*

Figure 4: Predictions for extrapolation of trained models in the heterogeneous jobs setup.

Table 1: R-squared metric for different models and evaluation scenarios.

	Homogeneous Jobs			Heterogeneous Jobs		
Observable	BiGRU	BiLSTM	Transformer	BiGRU	BiLSTM	Transformer
compute_time	-1.20	-1.08	-0.14	0.99	0.99	0.96
input_files_transfer_time	0.06	0.18	0.07	-0.27	-0.14	-0.46

5 Conclusion

We propose using surrogate modeling to evaluate the throughput of different infrastructure designs. We train three model architectures using simulator data to predict different job observables. From our evaluation results, the architecture choice does not significantly influence the accuracy of the predictions at the current stage of development. All three architectures decrease the execution times by orders of magnitude compared to DCSim.

At the current stage of the models inaccuracies are observed. We suspect a lack of input information given to the models as the predominant source. While the models are able to predict the compute times of our heterogeneous jobs scenario, where some implicit information about the infrastructure can be extracted by the model, they fail to predict the transfer time of input files due to not being aware of the data infrastructure setup that is more complex.

Future work can build on these results by incorporating platform information into the training data to improve the predictions. This will become essential to ensure that the model performs accurately on arbitrary workload mixes. Moreover, training on real-world data instead of simulator data could enhance the capabilities and applicability of the models, also providing valuable data in regimes where the simulation, due to its scaling behavior, is not able to feasibly produce large amounts of training data.

Acknowledgments

This work was supported by the pilot program Core Informatics of the Helmholtz Association (HGF) and by the German Federal Ministry of Education and Research (BMBF) project FIDIUM 05H21VKRC2.

References

[1] WLCG, The worldwide lhc computing grid, https://wlcg-public.web.cern.ch
[2] M. Horzela, H. Casanova et al., Modeling Distributed Computing Infrastructures for HEP Applications, EPJ Web of Conf. **295**, 04032 (2024). 10.1051/epjconf/202429504032
[3] M. Horzela, Measurement of Triple-Differential Z+Jet Cross Sections with the CMS Detector at 13 TeV and Modelling of Large-Scale Distributed Computing Systems (2023). 10.5445/IR/1000165566
[4] H. Casanova, A. Giersch et al., Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing **74**, 2899 (2014). 10.1016/j.jpdc.2014.06.008
[5] H. Casanova, R. Ferreira da Silva et al., Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH, Future Generation Computer Systems **112**, 162 (2020). 10.1016/j.future.2020.05.030
[6] P. Velho, L.M. Schnorr et al., On the Validity of Flow-Level Tcp Network Models for Grid and Cloud Simulations, ACM TOMACS **23** (2013). 10.1145/2517448
[7] P. Velho, A. Legrand, Accuracy Study and Improvement of Network Simulation in the SimGrid Framework (2009). 10.4108/ICST.SIMUTOOLS2009.5592
[8] K. Fujiwara, H. Casanova, Speed and Accuracy of Network Simulation in the SimGrid Framework (2007), https://dl.acm.org/doi/10.5555/1345263.1345279
[9] A. Lèbre, A. Legrand et al., Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API (2015). 10.1109/CCGrid.2015.134
[10] L. Stanisic, E. Agullo et al., Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers (2015). 10.1109/ICPADS.2015.67
[11] T. Cornebize, A. Legrand, F.C. Heinrich, Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study (2019). 10.1109/CLUSTER.2019.8891011
[12] H. Salehinejad et al., Recent Advances in Recurrent Neural Networks (2018). 10.48550/ARXIV.1801.01078
[13] Y. Duan, Y. Liu et al., Improved BIGRU Model and Its Application in Stock Price Forecasting, Electronics **12**, 2718 (2023). 10.3390/electronics12122718
[14] K.A. Althelaya, E.S.M. El-Alfy, S. Mohammed, Evaluation of bidirectional LSTM for short-and long-term stock market prediction (2018). 10.1109/IACS.2018.8355458
[15] N. Wu, B. Green et al., Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case (2020). 10.48550/ARXIV.2001.08317
[16] V. Zhyla, Performance modeling of distributed computing (2024). 10.5445/IR/1000175096