Enhancing Surrogate Model Usability for Optimisation Experts through Extended ML Support in EvoAl

Nils Leusmann*
Data Science Center - Bremen
Bremen, Germany
leusmann@uni-bremen.de

Christina Plump†
University of Bremen
Bremen, Germany
cplump@uni-bremen.de

Bernhard J. Berger
Hamburg University of Technology
Hamburg, Germany
bernhard.berger@tuhh.de

Rolf Drechsler‡
University of Bremen
Bremen, Germany
drechsler@uni-bremen.de

Abstract

Surrogate models are a crucial aspect in many real-world application optimisation problems. Thus, optimisation experts do not only have to focus on the optimisation problem at hand, but also work on selecting and configuring appropriate models. This may be challenging, when no extensive expertise in machine learning (ML) is present. To address this issue, we extend the open-source data science research tool EvoAl to better support optimisation practitioners in focusing on their core tasks. Specifically, we enhance EvoAl's machine learning language to accommodate the growing complexity of surrogate modelling workflows. Our contributions include expanding the range of available models, introducing runtime validation of model configurations, and enabling the integration of pre-trained models via ONNX. These improvements lower the barrier to applying advanced ML techniques in optimisation, thereby facilitating more efficient and robust surrogate-based optimisation in practical applications.

CCS Concepts

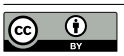
• **Theory of computation** → **Evolutionary algorithms**; • **Computing methodologies** → *Machine learning*; • **Information systems** → *Data exchange*; • **Software and its engineering** → *Domain specific languages.*

Keywords

evolutionary algorithm, surrogate model, domain-specific languages

ACM Reference Format:
Nils Leusmann, Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2026. Enhancing Surrogate Model Usability for Optimisation Experts through Extended ML Support in EvoAl. In *Genetic and Evolutionary Computation Conference (GECCO Companion '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3795101.3814674

*Also with University of Bremen.
†Also with Deutsches Forschungszentrum für Künstliche Intelligenz — Cyber-Physical Systems.
‡Also with Deutsches Forschungszentrum für Künstliche Intelligenz — Cyber-Physical Systems.



GECCO Companion '26, San Jose, Costa Rica

ACM ISBN 979-8-4007-2488-6/2026/07
https://doi.org/10.1145/3795101.3814674

1 Introduction

Surrogate models are frequently used in optimisation situations when the fitness function is either not available or the computation budget is too small. Surrogate models have been widely researched in the past decades [6, 9, 11] and are standard tool for optimisation in real-world application. Researchers with a focus on surrogate models may place many thoughts in how to build their model, which model management strategy to use, whether they have an online or an offline setting. In domain-based applications, however, the optimisation expert oftentimes is not necessarily an expert for surrogate models or even machine learning, nor may the time be given to become one during the project runtime.

When using machine learning models without in-depth knowledge, many (unwanted) errors can occur — from choosing models that do not fit the provided data, invalid hyperparameters for the given model or data, to wrongly computed goodness-of-fit measures which may be relevant for the model management strategy [7].

Equipping optimisation experts with the possibility to easily use machine learning capabilities within their optimisation framework and additionally ensure the correctness and sound choice of model and hyperparameter for their surrogate model can increase quality of resulting models and thus, optimisation results and subsequently let the optimisation expert focus on their core work — building the best optimisation possible for the problem at hand.

Our contribution is an improvement of the current machine learning features of EvoAl, covering three major aspects: (1) The rework of the machine-learning language, introducing a pipeline structure and enabling a fine-grained definition of training, and prediction. (2) The integration of several machine-learning algorithms into EvoAl, and the option to use externally trained models via an ONNX integration. (3) The addition of literature-based hyperparameter constraints ensuring a valid configuration setup.

2 Preliminaries

This section shortly explains an open-source optimisation research framework EvoAl and lists shortcomings in its machine learning capabilities that this work addresses.

2.1 The optimisation tool-suite EvoAl

EvoAl is an optimisation tool-suite for research and applied projects, whose primary goal is to explicitly describe the choice of algorithms and data processing via configuration files, rather than obscuring these algorithm-design decisions within the implementation. Besides optimisation, EvoAl covers machine-learning capabilities for creating and using surrogate models, as well as, data generation for

sound evaluations of optimisation algorithms. For describing these aspects, it offers several domain-specific languages (DSLs) [1], one for each use-case: Optimisation Language (`.ol`), Machine Learning Language (`.mll`) and Generator Language (`.generator`). EvoAl, however, shows its advantage through the use of a Data Description Language (`.ddl`), which allows specifying all domain data used in an optimisation problem. This information is, in turn, propagated and re-used in the aforementioned use-case specific DSLs, i.e., `.ol` files use references to `.ddl` files, clearly identify used data throughout the entire lifecycle, as well as include data information in the optimisation algorithm. Finally, EvoAl contains a fifth language, the Description Language (`.dl`), which can be thought of as a dictionary. Every parameter, operator, algorithm, encoding scheme is defined in here with its respective properties so that it can be correctly used in the use-case specific languages. This description language also allows the extension of EvoAl, when new algorithms or parameters are added. This language allows extending EvoAl without the need of changing the use-case specific DSLs.

2.2 Surrogate models shortcomings

EvoAl has provided machine learning for surrogate models from the start, however, the tool-suite's collaborators' focus has been mainly on optimisation [2]. Therefore, the machine learning aspects do suffer several shortcomings, what several research projects have shown in the last few years. One main drawback on the practice side is the restricted availability of ML-methods. EvoAl mainly offers support-vector regression as well as simple regressions. For modern ML-problems, this is usually not sufficient. Additionally, to use an ML-model for optimisation implies training it within the EvoAl landscape. This is a major drawback, as it prohibits the use of pre-trained models, generated by other ML frameworks. Furthermore, the structure of the ML-DSL is mainly directed at the use-case of full on-site training, which is consistent with the original use-case, but has drawbacks, when goodness-of-fit measures need to be recomputed or external models need to be used.

3 Method

This section describes the employed methods and how they support to achieve the contributions.

3.1 Machine learning language restructuring

We mainly address two issues regarding the machine-learning language to overcome some of the discussed shortcomings. First, we introduce the possibility to support several use-cases of machine-learning tasks. Second, we restructure the language to follow a pipeline concept which allows the re-use of common steps as well as default settings which simplify the use for non-experts while at the same time allowing detailed adjustments for experts.

Use-cases The most common use-cases in ML are `training`, `prediction`, and `testing`. Figure 1 shows the required steps for these use-cases. The `training` use-case needs training data as input, may potentially apply some preparation tasks to these data, i.e., perform a data validation (does the training data look as expected for these input features) or preprocess data in a specified manner (e.g., data normalisation), and finally provide well-curated training

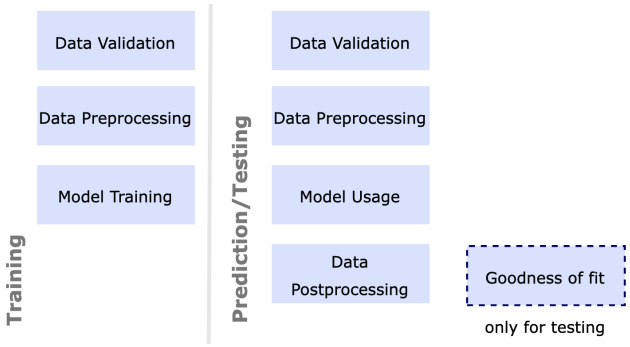


Figure 1: Use-case overview

data, a chosen model, and hyperparameters to the actual model learner. The learned model is then serialised for later usage.

`Prediction` is the main use-case for surrogate-assisted optimisation. For this, data preparation may take place, c.f. last use-case. Afterwards, the learned model is applied to a data instance, the resulting data is potentially post-processed (e.g., revert the normalisation) for further usage.

`Testing` works very similar to `Prediction`, as performing many predictions is the core task of `testing`. Thus, `Testing` contains a looped `prediction` pipeline (or vectorised, depending on one's view), which predicts the result of all testing data, performing all steps of the previously explained pipeline. Finally, it computes a goodness-of-fit measure, e.g., a root-mean squared error.

Concepts like cross-validation are modelled by employing the training and testing use-case in reiteration.

Language Structure The goal was to adapt the language structure such that these use-cases can (a) be modelled, (b) re-use identical parts, (c) separate general information from pipeline definition.

The new MLL keeps the old import block which allows the `.mll` to be informed about definitions in `.ddl`-files as well as `.dl`-files. The remainder of a `.mll`-file is divided into two parts: First, the specification of the different building blocks of the use-cases, e.g., the definition of a learning task, or the definition of data validation. This section also includes the specification of the *learning-task*, containing all necessary information, regarding files to read data, definition of data, output information and more. Listings 1 shows an exemplary specification of this block. The second section contains optional *use-case* defintions. Every *use-case* has a default implementation which is used when this section does not specify any changes. ML experts, however, are free to adapt the use-case behaviour with this section. We will describe the training/testing use-case in greater detail, more extensive examples can be found in the repository[1].

```
specify learning-task {
  input features 'x0', 'x1'
  output features 'y0'

  learning from ["data.json"]
  testing with ["dataTesting.json"]
  serialise to "knn.pson"
}
```

Listing 1: Examplary learning task definition

[1]Visit https://www.evoal.de and navigate to GitLab

Testing Use-case A second part of the first building block definition is the specification of the goodness-of-fit measures, which are used in the testing use-case. Listing 2 shows an example that configures two goodness-of-fit measure functions to call.

```
specify gof begin
  rmse();
  R^2();
end
```

Listing 2: Two configured goodness-of-fit measures

These building blocks are then combined to a learning use-case definition in the second section of the file. A possible learning use-case is shown in Listing 3. First, the training data is loaded (the information on the training data is specified in the learning task), then the validation and preprocessing pipelines (also specified in the first section) are called. Afterwards, the model learner is called, which trains the ML model. Now, the user is free to add, e.g., a goodness-of-fit calculator, which calculates the configured goodness-of-fit measures, before the model is stored.

```
specify learning begin
    execute([
      step {
        component 'training-data-loader' {}
      },
      pipeline 'validation',
      pipeline 'preprocessing',
      step {
        component 'model-learner' {}
      }
    ]);

    'gof-calculator'();
    'store-model'();
end;
```

Listing 3: Examplary learning usecase definition

3.2 Extension of ML-models

3.2.1 Additional models integration. The last EvoAl release[2] includes a limited set of ML-capabilities, mainly linear regression and support vector regression. While they provide core capabilities for using surrogate models, they do not allow optimisation experts to gain the full variety and depth of surrogate models. We integrated several ML-techniques from the smile-library into EvoAl [10] to allow optimisation experts to choose more freely between models to build their surrogate models. The most important are random forest [3], k-nearest neighbour [5], and Gaussian processes [12].

3.2.2 Open Neural Network eXchange. In addition, to the new learning models, EvoAl has been extended to support models stored in the ONNX format. ONNX—which stands for open neural network exchange (ONNX) format—is a bytecode-like file format to store ML models. The format has seen a steep rise in popularity, especially as an inter-exchange tool [8]. After training an ML model, supporting libraries, such as PyTorch and others, can use ONNX as a serialization format that can be loaded and executed in any programming language that supports a ONNX runtime. Most popular programming languages support an ONNX runtime, and there are also libraries to translate ONNX to CUDA, to allow the execution of such models on GPUs. This makes ONNX it a great choice as exchange format for heterogeneous applications that use different programming languages for training and ML model usage a key factor is that this allows users to use pre-trained models (from previous research, or provided by project partners from the domain) instead of having to train themselves. Additionally, it allows models based on neural networks, which is a main machine learning technique that had not been introduced to EvoAl so far.

[2]Available at https://zenodo.org/records/17494875

3.3 Introducing Hyperparameter Constraints

Many machine learning models depend on hyperparameters. For k-nearest neighbour, a k has to be determined, support vector regression needs kernel parameters and tolerance values, random forest a branching coefficient, tree depth and more. These hyperparameters are set before training, usually with respect to knowledge of data and the model. However, a lot of expertise is necessary to choose hyperparameters that yield well-trained models. Oftentimes, the choice is also done via hyperparameter optimisation.

Either way, it is important that valid hyperparameters are chosen. We therefore propose a hyperparameter validation that validates whether allowed hyperparameters are chosen (with respect to the chosen model and the training data). To that end, we conducted an extensive literature research for machine learning method hyperparameter constraints. These were compared and condensed to a set of constraints for each hyperparameter of each included machine learning method in EvoAl. We extended the `.dl` language to be able to capture constraints for the parameters defined within, using `@LowerBoundary()` and `@UpperBoundary()` for constraining the hyperparameter range, and multivariate constraints for introducing dependencies between several hyperparameters. For example, as a $k <= 0$ does not make much sense in a k-nearest neighbour models, Listing 4 depicts how this lower boundary is realised.

```
type knn extends surrogate {
  /** The number of neighbours to take into account. */
  @LowerBoundary{ inclusive := true; boundary := 1; } 'k' : int;
}
```

Listing 4: Defining boundaries for hyperparameters

When an `.mll` is used in running a use case, the hyperparameters are validated against the constraints. If they are violated, an error is issued, as well as an explanation for the error displayed and a suggestion for a different choice. The user does now have the choice to adapt the hyperparameter and try again.

4 Implementation

Restructured language We adjusted the original grammar to reflect our proposed changes. Xtext [4] generates all necessary artefacts to parse and validate the DSL files and to create EMF [13] model instances of the file contents.

ML model extension One core principle of EvoAl is modularity and extensibility supporting us in adding the features proposed in 3.2. Additional ML models were implemented using the Smile library [10] and are part of EvoAl's smile-surrogate plugin. Following the EvoAl-API, they are an extension of the base class *ModelFunction*. We also implemented the model learner interface for supporting these models in the learning process. We are using the Java implementation—called ONNX runtime—to load and integrate ONNX-serialised ML models into EvoAl. We implemented an ONNX-specific *ModelFunction* and *ModelStorage* class. If configured,

the model storage class loads the serialised ML model and creates the ONNX model function that EvoAl uses for prediction. During the load process, the in- and output features of the ONNX model are mapped to the data description given in EvoAl's data description language. The ONNX model function maps EvoAl's data structures to the in- and output features of the ONNX runtime. This is done by iterating over the input parameters and converting them from EMF objects to *ONNX-Tensors*.

Hyperparameter constraints EvoAl has a built-in mechanism for validating algorithm input during runtime. We adapted this mechanism to include hyperparameter validation. When loading a DSL file, EvoAl links the used elements to the internal algorithm definitions. These definitions include the expected parameters, their types, and the described constraints on these parameters. The validation component—comparable to an interpreter— checks if the specified parameters adhere to the given constraints. In case of violations, EvoAl provides feedback on the violation. This mechanism ensures the validity of configured ML models and can be used for training and prediction without running into runtime errors or invalid models due to misconfiguration.

5 Discussion

Related Tools There are many different opimisation and ML tools, but few cover both topics. On the one hand, *sklearn* [3], *pytorch*[4], or *tensorflow*[5] focus primarily on machine learning. These frameworks are quite popular and support ML tasks, but have very few (if any) optimisation functionalities. On the other hand, *scipy.optimize*[6] or the MOEA Framework[7] focus on optimisation, but have less focus on ML. Recent development within the MOEA framework seems to work in similar fields. For example, they recently added new constraint classes, such as *Minimize*, and expanded their repertoire of learning algorithms with k-means. All of these tools, however, do not separate algorithmic design decisions and source code, which is unique to EvoAl. Using EvoAl (within the provided algorithms) does not require programming expertise, which allows either experts to focus on their expertise, or domain experts with little to no programming expertise to use these functionalities at all.

Future Work We will focus on improving the following ML-related features in EvoAl. First, we continue the integration of ONNX models, by adding support for serializing EvoAl-trained models to ONNX files. Additionally, we plan to extend the current ONNX (de-)serialisation by the support of model meta-information, such as goodness-of-fit measures. In terms of extending ML-capabilities, classification problems are at the top of our list, as well as online ML-model adaption. We might consider improved functionalities for integrated optimisation under surrogate models, which is not yet present. We plan to integrate prediction intervals as goodness-of-fit measure for all present machine-learning methods, which will require the integration of bootstrapping functionality. Finally, considering quality of models, we would like to integrate hyperparameter optimisation (using EvoAl's optimisation capabilities) and integrate visualisation of the training process.

[3]Available online at https://scikit-learn.org/stable/
[4]Available online at https://docs.pytorch.org/docs/stable/index.html
[5]Available online at https://www.tensorflow.org/api_docs
[6]Available online at https://docs.scipy.org/doc/scipy/tutorial/optimize.html
[7]Available online at http://moeaframework.org

6 Conclusion

Surrogate models are an integral part of real-world applications of evolutionary algorithms. Enabling optimisation experts to use machine-learning models (either pre-trained, or self-trained) easily without having to worry about a valid choice of hyperparameters is a relevant feature for an optimisation tool suite. We presented an extension to the EvoAl optimisation tool suite that allows a validated integration of ML capabilities. The main philosophy of EvoAl, separating implementation from algorithmic-design decisions, and enforcing valid configuration is upheld by integrating hyperparameter constraints. Additionally, the ML capabilities are greatly enhanced by adding more ML models and allowing the usage of pre-trained models in the ONNX format. Usability is increased through the restructured language concept, which now explicitly focuses on the key ML use cases. All in all, our contribution enables the optimisation tool suite EvoAl to include more variety, flexibility, and quality into surrogate-assisted optimisation problems. We plan to extend our work by adding more goodness-of-fit measures, as well as hyperparameter optimisation within the suite itself.

Acknowledgments

We thank the anonymous reviewers for their insights and improvement suggestions. This work was supported by the German Federal Ministry of Research, Technology, and Space within the INDIFUN-AI project under grant number FKZ 03F0975B.

References

[1] Bernhard J Berger, Christina Plump, and Rolf Drechsler. 2023. EVOAL: a domain-specific language-based approach to optimisation. In *2023 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–10.
[2] Bernhard Johannes Berger, Christina Plump, Lauren Paul, and Rolf Drechsler. 2024. EvoAl-codeless domain-optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1640–1648.
[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[4] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: Implement Your Language Faster than the Quick and Dirty Way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion* (Reno/Tahoe, Nevada, USA) *(OOPSLA '10)*. Association for Computing Machinery, New York, NY, USA, 307–309. https://doi.org/10.1145/1869542.1869625
[5] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.
[6] Chunlin He, Yong Zhang, Dunwei Gong, and Xinfang Ji. 2023. A review of surrogate-assisted evolutionary algorithms for expensive optimization problems. *Expert Systems with Applications* 217 (2023), 119495. https://doi.org/10.1016/j.eswa.2022.119495
[7] Károly Héberger. 2024. Frequent Errors in Modeling by Machine Learning: A Prototype Case of Predicting the Timely Evolution of COVID-19 Pandemic. *Algorithms* 17, 1 (2024). https://doi.org/10.3390/a17010043
[8] Purvish Jajal, Wenxin Jiang, Arav Tewari, Erik Kocinare, Joseph Woo, Anusha Sarraf, Yung-Hsiang Lu, George K Thiruvathukal, and James C Davis. 2023. Analysis of failures and risks in deep learning model converters: A case study in the onnx ecosystem. *arXiv preprint arXiv:2303.17708* (2023).
[9] Yaochu Jin. 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (2011), 61–70. https://doi.org/10.1016/j.swevo.2011.05.001
[10] Haifeng Li. 2014. Smile. https://haifengl.github.io.
[11] Linqiang Pan, Cheng He, Ye Tian, Handing Wang, Xingyi Zhang, and Yaochu Jin. 2019. A Classification-Based Surrogate-Assisted Evolutionary Algorithm for Expensive Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 23, 1 (2019), 74–88. https://doi.org/10.1109/TEVC.2018.2802784
[12] Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Summer school on machine learning*. Springer, 63–71.
[13] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2008. *EMF: Eclipse Modeling Framework* (2nd ed.). Pearson International.