The International Journal of Advanced Manufacturing Technology (2021) 115:3513–3532
https://doi.org/10.1007/s00170-021-07248-3

ORIGINAL ARTICLE

Cognitive capabilities for the CAAI in cyber-physical production systems

Jan Strohschein[1] · Andreas Fischbach[2] · Andreas Bunte[3] · Heide Faeskorn-Woyke[1] · Natalia Moriz[3] · Thomas Bartz-Beielstein[2]

Received: 21 December 2020 / Accepted: 5 May 2021 / Published online: 8 June 2021

Abstract

This paper presents the cognitive module of the Cognitive Architecture for Artificial Intelligence (CAAI) in cyber-physical production systems (CPPS). The goal of this architecture is to reduce the implementation effort of artificial intelligence (AI) algorithms in CPPS. Declarative user goals and the provided algorithm-knowledge base allow the dynamic pipeline orchestration and configuration. A big data platform (BDP) instantiates the pipelines and monitors the CPPS performance for further evaluation through the cognitive module. Thus, the cognitive module is able to select feasible and robust configurations for process pipelines in varying use cases. Furthermore, it automatically adapts the models and algorithms based on model quality and resource consumption. The cognitive module also instantiates additional pipelines to evaluate algorithms from different classes on test functions. CAAI relies on well-defined interfaces to enable the integration of additional modules and reduce implementation effort. Finally, an implementation based on Docker, Kubernetes, and Kafka for the virtualization and orchestration of the individual modules and as messaging technology for module communication is used to evaluate a real-world use case.

Keywords Cognition · Industry 4.0 · Big data platform · Machine learning · CPPS · Optimization · Algorithm selection · Simulation

✉ Jan Strohschein
jan.strohschein@th-koeln.de

Andreas Fischbach
andreas.fischbach@th-koeln.de

Andreas Bunte
andreas.bunte@th-owl.de

Heide Faeskorn-Woyke
heide.faeskorn-woyke@th-koeln.de

Natalia Moriz
natalia.moriz@th-owl.de

Thomas Bartz-Beielstein
thomas.bartz-beielstein@th-koeln.de

[1] TH Köln, Institute of Computer Science, Gummersbach, Germany

[2] TH Köln, Institute for Data Science, Engineering, and Analytics, Gummersbach, Germany

[3] Institute Industrial IT, OWL University of Applied Sciences and Arts, Lemgo, Germany

1 Introduction

Artificial intelligence (AI) in cyber-physical production systems (CPPS) can help to significantly reduce costs [22], but its implementation requires expert knowledge, and thus might be cost-intensive [15]. To tackle these challenges, a modular and extendable Cognitive Architecture for Artificial Intelligence (CAAI) in CPPS was introduced in previous work [15]. The proposed architecture can be used by companies of all sizes, even SMEs, as the implementation is possible on an existing computing cluster, smaller edge devices, and also the cloud. Our current implementation is focused on optimization use cases and provides an OPC UA connector to connect to production machinery and other building blocks to adjust the architecture to a specific use case without much implementation effort, e.g., additional algorithms if required for an application scenario. The open-source distribution of the architecture and its implementation assist users with examples. The architecture defines process pipelines as a sequence of processing modules, e.g., a preprocessing module, followed by a modeling module wrapped up by an optimization module to find an optimal configuration of the

model. The usage of cognitive capabilities for the selection of AI algorithms enables the system to learn over time and choose suitable algorithms automatically and thus replace the expert knowledge, to some extent. This can be a key feature to reach a high degree of autonomy and efficiency in CPPS [7].

The work at hand describes the cognitive module and its functionality, which is further referred to as *Cognition*, in more detail than [15]. The task of the *Cognition* is to propose candidate pipelines with proper parameters, compute several pipelines in parallel, evaluate the pipeline quality, and switch to promising pipelines during the operational phase, e.g., if they are likely more efficient w. r. t. accuracy or resource consumption. Furthermore, the *Cognition* does not require deeper AI knowledge from the user.

This paper focuses on the optimization use case, whereas the general concept will be extended to other use cases, such as Condition Monitoring, Predictive Maintenance, or Diagnosis. The main contributions of this paper are:

- Automatic algorithm selection in dynamic environments,
- Automatic creation of machine learning pipelines, based on selected algorithms, and
- Real-world evaluation of the cognitive module and available CAAI implementation for the use case on GitHub.

The remainder of this paper is organized as follows: Section 2 provides an overview of related works. The concept of the cognitive module is described in Section 3 along with components that are closely related and important for its behavior. Details about the implementation on the CAAI big data platform (BDP) and an evaluation on a real-world use case can be found in Section 4. Finally, Section 5 discusses our major findings and resulting future research tasks.

2 Related work

The main contributions of our work concern several research areas. Thus, the first sub-section considers architectures for application of AI in CPPS. The second sub-section reviews the orchestration and scheduling of machine learning workloads on Kubernetes, an open-source system for automating deployment, scaling, and management of containerized applications. Details on Kubernetes can be found in [8]. The third sub-section addresses the algorithm selection and tuning to generate feasible optimization pipelines.

2.1 Architectures for AI in CPPS

An architecture supports the design of complex software systems. It provides implementation guidance and best practices based on proven concepts within a domain. The result is a blueprint for a domain-specific class of systems and facilitates communication and knowledge transfer. We evaluated proposed reference architectures for the application of AI in CPPS from the field of automation and cognitive sciences in previous work [7] and proposed CAAI [15] as an alternative that was more focused on software development for the application of AI in CPPS. Thus, we build on previous work, e.g., [46] and [19], to develop a three-tier architecture that enables an adaptable system with rapidly changing configurations. CAAI implements algorithms as microservices that communicate via messages on different bus systems. Furthermore, the architecture provides building blocks to easily integrate databases and APIs. Those can be accessed over the local network or the Internet to provide up-to-date information and allow to adjust settings for the platform or individual algorithms during run time. CAAI uses Kubernetes to orchestrate the necessary services and algorithms for online processing of the production data to give near-real-time insights into the production process.

2.2 Orchestration and scheduling on the big data platform

The CAAI uses Kubernetes for the orchestration of machine learning pipelines. Research found several projects that build machine learning workflows on top of Kubernetes.

Altintas et al. present Chase CI as a highly scalable infrastructure project for machine learning based on Kubernetes [1]. They build a cluster consisting of resources from 20 partner institutions and portray a deep learning use case where neural networks learn from weather data. A list of steps is developed called *Process for the Practice of Data Science* to guide AI experts. The work shows no signs of cognitive capabilities and is not easy to extend, as there are no templates or abstractions.

Subramaniam et al. propose abstractions for machine learning workloads in Kubernetes [44]. They develop a set of Custom Resource Definitions (CRDs) and custom controllers for Kubernetes to make it easier for an AI expert to create new machine learning jobs on Kubernetes. The work does not possess cognitive abilities or help the user to select a suitable algorithm.

A related project that stems from industry is Kubeflow. It is meant to simplify the process of deploying machine learning workflows on Kubernetes and started as an internal

Google TensorFlow framework that was open-sourced in late 2017. In Kubeflow, a pipeline describes a single machine learning workflow, where each component is packaged as a Docker image. Those pipelines can be created programmatically with a domain-specific language (DSL) they provide. It is also possible to convert Jupyter notebooks into a pipeline via a graphical-user-interface and a plugin [37]. Even though the project provides a nice graphical interface, it is targeted mostly towards data scientists and machine learning engineers. Right now, the project allows to process data in batches and has no built-in streaming capabilities. The user also has to build the pipeline by himself and Kubeflow does not support the user to choose the right algorithm for a use case.

Our cognitive module dynamically schedules pipelines on Kubernetes based on the available system resources. Several other works consider the scheduling of machine learning workloads on a Kubernetes cluster. The researchers [10, 24, 35] use insights into cluster information and job states for dynamic allocation with a focus on energy efficiency and minimization of training times. Peng et al. developed a schedular that is capable of updating the resources based on training speed and predicted time needed for model convergence [35]. The related works improve the performance compared to the standard Kubernetes scheduler or traditional cluster schedulers, e.g., Mesos and Yarn. They use the system resources efficiently but do not determine which jobs should be instantiated to solve a machine learning problem. While those related works target machine learning on Kubernetes, we found no projects that specifically target machine learning for CPPS in Kubernetes.

2.3 Algorithm selection and tuning

Muñoz et al. [33] provide a survey on methods to address the problem of algorithm selection for black-box continuous optimization problems. They precisely describe the algorithm selection problem (ASP) as challenging due to the limited theoretical understanding of the strengths and weaknesses of most algorithms. Furthermore, due to a large number of available algorithms, it is difficult or impossible to overview all algorithms. A framework for ASP was introduced by Rice [39] and described more recently by Smith-Miles [42] as a four-step process. However, the implementation is still challenging and often interpreted as a learning task [42]. Typically, classification or regression models are employed to address this task [33]. Both methods come with some disadvantages, e.g., classification methods have to be re-trained if there are changes in the algorithm, whereas regression models are modular but with its higher number of elements, regression models are prone to failures [33].

One way to extract features of the problem instances is provided by Exploratory Landscape Analysis (ELA) [29]. ELA characterizes the problem by a larger number of numerical feature values that are grouped into categories. These features are determined by numerical computations based on sampling of the decision space. The advantage of these so-called *low-level features* is that they can be determined automatically, although they are related to some high-level features, whose creation requires knowledge. Based on these features, the selection of an optimizer process can be performed, e.g., by using a classifier [29]. A combined approach of machine learning (ML) and the application of ELA, where results from previous Black-Box-Optimization-Benchmarking workshops taking place at the Genetic and Evolutionary Computation Conference (GECCO) were used to train the selectors, can be found in [25]. This approach currently comes with some drawbacks. Recent research empirically shows that not all ELA features are invariant to rotation, translation (shifting), and scaling of the problem [27]. Additionally, features are sensitive to the sampling strategy employed [38].

Automated machine learning (AutoML) [12, 13] and hyperheuristics can choose and configure a suitable algorithm automatically. That includes steps such as data preprocessing, algorithm selection, and hyperparameter optimization (tuning) [17, 34, 45]. However, these approaches are built to process offline data, so they expect a training and a test dataset. In comparison to that, our approach has to deal with online data, so the dataset is continuously growing and not available a priori. Especially in the first few production cycles, the amount of available data is not sufficient to partition the dataset into test and training data as AutoML methods would require.

For the evaluation of the method performance on real-world problems, or, more precisely, real-world–related test problems, Zaeffer et al. employed Gaussian process simulation for discrete optimization [52]. This extends a premature work of varying Gaussian process model parameters within a certain range to retrieve instances from a problem class [16]. This enables the generalization of performance evaluation methods on problem classes, which is described in [14]. The application of Gaussian process simulation for the continuous domain can be found in [53].

The approach to tackle the ASP in this paper is based on the objective function simulation employing an initial design sampling of the process. Conducting small-scale benchmark experiments determines feasible and well-performing algorithms from an available portfolio in an iterative process.

3 Concept

The concept of the cognitive module and its implementation is presented in this section. In the first subsection, the general concept of the CAAI and the cognitive module is introduced. Then, a concept for defining declarative goals for the CAAI is presented. In the following subsection, the description of algorithms is addressed, which is stored in the knowledge base and acts as a basis for the selection of algorithms. A detailed description of the cognitive module follows. Finally, the connection between the *Cognition* and the other modules is described, since the cognition has to configure the different modules automatically.

3.1 General concept of CAAI

The cognitive architecture CAAI builds upon the idea of modeling the information, data applications, and streams required for specific tasks in the I4.0 scenario while providing reliability, flexibility, generalizability, and adaptability. The concept, depicted in Fig. 1, is based on a three-tier architecture to simplify interoperability and ensure horizontal scalability. The CAAI-BDP, depicted in dark gray, wraps the architecture and arranges software modules in two processing layers (shaded in light gray), the Data Processing Layer (DPL), and the Conceptual Layer (CL). The layers are connected via bus systems (data, analytics, and knowledge bus), which are colored in blue. Arrows demonstrate the designated information flow.

Data from a CPPS enters the system at the very bottom. The *Protocol Translation* transforms incoming data and sends it to the data bus. The pre-processing module receives the raw data, performs the necessary steps to clean the data, and publishes the results back to the data bus. Other modules in the DPL, such as *Modeling* or *Optimization*, utilize data from the data bus and transfer their analytical results onto the analytics bus. Modules in the CL process information about the user-defined aims and the business logic for a given use case. They evaluate the results from the analytics bus, determine the parameters to adjust the CPPS via the adaption module, and measure the overall system performance and available resources through the monitoring module. The CL modules also interact with the knowledge bus to generate reports for the user and to process new instructions. The human-machine interface (HMI) communicates with the CAAI BDP through the knowledge bus, where the user can add new declarative goals during operation or adapt the knowledge base. Data and information are transported by messaging within the CAAI BDP. To ensure the correct interpretation in different modules, schemas are defined for different types of data. However, the central element of the architecture is the *Cognition*, which selects, orchestrates, and evaluates different algorithms, depending on the use case. Schemas

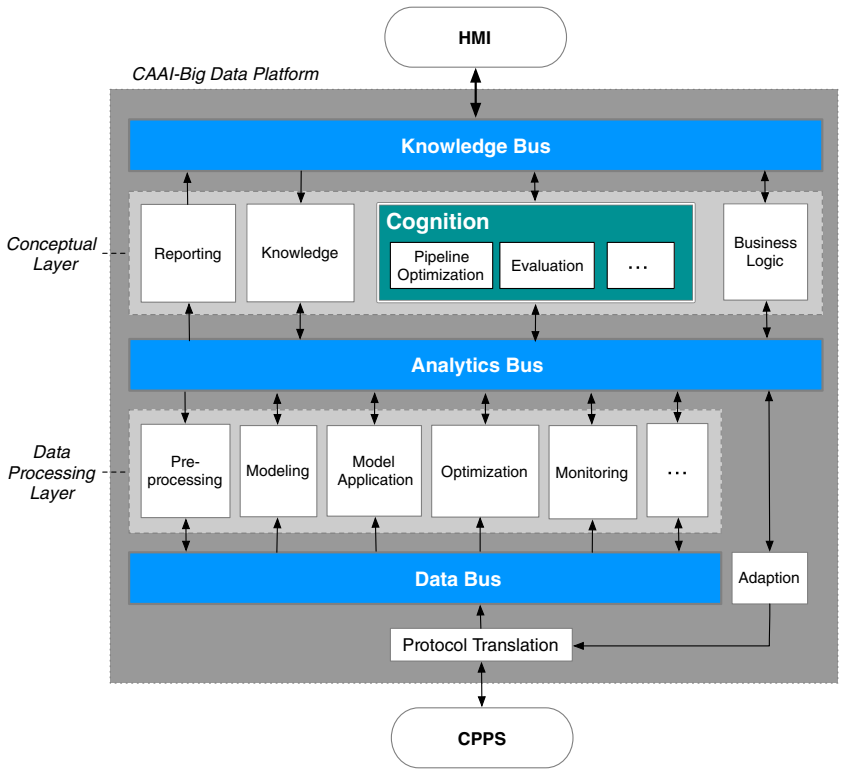

Fig. 1 CAAI Architecture, consisting of a CAAI-Big Data Platform, three bus systems, a conceptual layer, and a data processing layer

are essential for the Cognition, to identify compatible modules during the algorithm selection process.

Different algorithm types have to be considered to choose a promising candidate for the current state of the CPPS. For example, regression models can be learned with only a few data points. Still, they typically require expert knowledge, i.e., about the type of dependencies, and they are typically less accurate compared to neural networks [23, 47]. Neural networks may also have some disadvantages, as they typically need more training data and more resources for the learning process. Pre-trained nets [51], such as in computer vision, are not always available for versatile systems such as CPPS. Due to the modular structure of the CAAI, new algorithms can be integrated easily, e.g., to incorporate more accurate pre-trained models or when different model techniques come into focus for the problem at hand. However, knowledge about the algorithms has to be made available for the *Cognition*, so it can decide which is the most promising algorithm in a particular state. Because the operational states and thus the best choice for an algorithm will change over time, the composition of active modules and their communication over the bus system will change during run time. Providing a pre-defined set of modules and the capability to add new modules reduce the overall implementation complexity by building a cohesive yet modular solution. In the work at hand, the *Cognition* is described in detail. Furthermore, implementation and evaluation of the cognitive module are performed in this paper.

3.2 Declarative goals

To enable easy usage of the CAAI, the method to optimize the CPPS will be selected automatically. This is possible due to CAAI utilizing declarative goals, e.g, the user specifies the goal, not the single process steps to achieve it. However, this declarative goal must be formulated at least once and goals such as *Resource Optimization* are too unspecific and cannot be converted into an appropriate pipeline. Thus, CAAI implemented a step-wise procedure to assist the practitioner in the process.

The task of an algorithm in the context of an optimization problem is to find the setting of one or more control parameters $x \in \mathbb{R}^n$ which minimizes (or maximizes) a function $y = f(x)$ subject to constraints $\Phi(x)$:

$$\operatorname{argmin} f(x) \quad s.t. \quad \Phi(x)$$

The formulation of an optimization problem typically involves three steps [5]:

1. Selecting control variables x,
2. Choosing the objective function, and
3. Identify constraints on x.

Inside CAAI, the first and third steps result in the definition of the parameters that control the process and adapt the CPPS via the adaption module (see Fig. 1). The *Business Logic* observes the parameter constraints and verifies that all values are feasible before the *Adaption* sends adjusted parameters to the CPPS. For the second step, the formulation of the objective function, we developed a multi-stage goal selection, which guides machine operators through the goal selection. This is presented through the four-stage selection for optimization. In the first stage, the user selects the overall goal, such as optimization, anomaly detection, condition monitoring, or predictive maintenance. In the second stage, the signals are selected. This could be all, many, or just a single signal, depending on the overall goal and the use case. In the third stage, aggregation functions can be selected, such as mean, delta, min, or max value, or the value itself. In the last stage, the user selects the optimization goal, e.g., minimizing or maximizing. With this four-step process, the user can select the optimization goals on a more abstract level. This is shown in Fig. 2, on an energy optimization use case of a bakery, which is described in more detail in [7]. The goal of the use case is the minimization of the peak power consumption. Therefore, the user chooses *Optimization*, the *power signal*, the *maximum value* of the signal, and *minimize*. As one can see, the usage of such applications is quite straightforward.

It is possible to select multiple overall goals, e.g., optimization and anomaly detection, where each goal results in a separate pipeline. If the goal optimization is selected multiple times, a single criteria optimization algorithm is used in this work, since we do not have multi-criteria optimization algorithms in the portfolio until now. The different optimization goals are normalized and charged with user-defined weights. So, this approach enables the user to select also complex goals in an easy way.

3.3 Algorithm characteristics

Algorithms that are available in CAAI are described in the knowledge module using a common schema to enable the cognitive module to select suitable algorithms. From a knowledge representation perspective, these are frames,

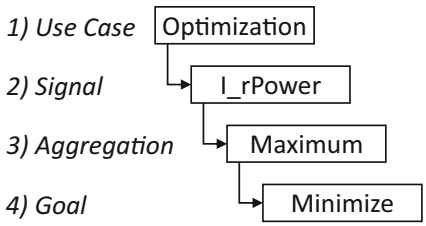

Fig. 2 Application of the four-stage declarative goal selection

which were introduced by Marvin Minsky in 1974 [30]. A frame represents an instance (in this case an algorithm) using a pre-defined schema; and thus, each slot of a frame can be interpreted semantically. Furthermore, relations between frames can be defined, which is in our case the connection to a declarative goal. The schema that was developed for this work is shown in Table 1. The presented properties can be divided into mandatory properties that must have a certain value to enable algorithm usage, such as the property *use case class*. Other properties represent just the performance of the algorithms; and thus, the best matching algorithms can be chosen, such as the property *performance*. The required *input data type*, *output data type*, and the *use case class* are mandatory properties and enable determining which algorithms can be used for a certain pipeline. The property *algorithm class* is used as a high-level property for the selection process.

According to the *No free lunch theorem* [49], we assume, that there is no single optimizer, which performs best on all optimization problems. Therefore, the CAAI provides several optimizers from different optimization algorithm classes with different features, operators, and parameters. Most likely, algorithms from different classes will perform differently on problems according to both the best-found value and also resource consumption. It might be beneficial, to ensure the test of a broad range of algorithms w. r. t. the available resources. Furthermore, it can be learned that a certain class of algorithms is suitable or unsuitable for a certain CPPS. The listed algorithm classes (see Table 1) are selected based on a taxonomy by Stork et al. [43]. In an increasing order, the given classes *Hill-climber*, *Trajectory*, *Population*, and *Surrogates* arguably reflect the complexity of its members. In the following, we shortly introduce the different algorithms classes.

Hill-climber These algorithms are more or less greedy algorithms, especially useful for local search and often used in combination with more sophisticated algorithms (e.g., Surrogate-based optimizers). A standalone application can be highly efficient if the problem turns out to be unimodal. An example for this class is the L-BFGS-B algorithm [28].

Trajectory Algorithms from this class can escape local optima to avoid premature convergence and are needed if the problem is multimodal. Famous members of this class are, e.g., Simulated annealing [26], one of the first global optimization algorithms, and Tabu search [18].

Population Population-based algorithms can be further divided into collaborative and competitive algorithms. An example for collaborative algorithms is particle swarm optimization, where a swarm of individual solutions coordinates their search by exchanging information of their current state. Competitive strategies are implemented by, e.g., evolutionary algorithms (EAs), which imitate evolutionary development of populations of individuals (candidate solutions) over several generations, applying random mutations and crossover operations to create the offspring population according to the given fitness of the individuals. A survey of particle swarm optimization (PSO) variants and comparison of their performance with EAs on different manufacturing problems can be found in [41]. Algorithms from this class are sensitive to their parameters (e.g., the population size, crossover and mutation rate, number of iterations). These parameter settings control, e.g., the balance between exploration and exploitation and thus the convergence speed. For an insight into the utilization of different strategies to optimize metaparameters (tuning) and performance analysis, we refer to [11]. A recent example showing the importance of parameter tuning of PSO for a real-world manufacturing problem can be found in [40].

Surrogate This class of optimizers in our approach uses a surrogate model (e.g., a regression model) as a replacement for the (potentially costly) objective function to estimate the objective function (via model predictions) relatively cheap (in comparison to the objective function itself). Then, the surrogate model is searched for the optimum and the found candidate(s) is evaluated on the objective function. The retrieved values are used to update the model in the next iteration.

As the efficiency of many algorithms relies on proper parameter selection, tuning is highly recommended to reveal the full potential of the algorithms' performances on the underlying problem class. The usage of a data-driven generation of test instances combined with a randomized selection allows automatic tuning of algorithms without data scientist knowledge. This approach can tackle the possible issue of overfitting algorithms on fixed and not relevant problems [14].

As the application of the CAAI tries to minimize the required knowledge regarding data science and algorithmic concerns, and the occurring use cases, i.e., the problem class and its features can be considered a priori unknown, it is beneficial to have several different algorithms with a broad variety of implemented concepts available. This should increase the possibilities of an algorithm selection component like the cognitive module to select an efficient algorithm according to the problem.

Three properties are specified to an assigned machine. The values are *Performance*, *computational effort*, and *RAM usage* and are updated by the *Cognition* during the evaluation process. An algorithm may not be suitable for a specific use case, even if the properties indicate a good

match. However, with the dynamic parameters, the CAAI is able to overcome this challenge and skip those algorithms.

3.4 Cognitive module

The purpose of the *Cognition* is to select and parameterize suitable algorithms for a given use case. The selection is based on four kinds of information: (i) the use case and knowledge about suitable algorithms to reach the goal; (ii) specifications of the data, such as available data, and type of data; (iii) characteristics of the algorithms, such as run time or memory consumption; (iv) experience about the performance of the algorithms in previous applications.

The problem is given by the user, as described in Section 3.2. Information about the algorithms is stored in the knowledge module. Every algorithm has a signature that describes the data required by the algorithms and the data they provide. Based on this information, possible pipelines can be identified and rated regarding the expectations to be suitable to solve the problem. The rating is performed based on the three characteristics mentioned above ((ii)–(iv)).

Typically, algorithms have some requirements, e.g., regarding the number of needed training data, or data types. These are hard criteria, which do not allow the usage of the algorithm for the type of system if they are not fulfilled. However, since the amount of data is continuously growing during production cycles, the algorithm may become feasible at a later point in time. These criteria are checked frequently by the cognitive module. If the performance is dramatically decreasing or stagnating over a number of production cycles, perhaps due to premature convergence of an algorithm, the selected pipeline can be changed in between.

Depending on the available resources, the dimension of the data, and the amount of available data, the computational effort of each pipeline can be rated. Since resources are limited, it is required to consider whether one pipeline with expected good results but high resource usage or several pipelines with a lower chance of good results should be evaluated. In an extreme case, it might be impossible to apply a certain pipeline, if the resources are very limited. So the resources are major criteria for the selection of suitable pipelines.

To rate the quality of results for a certain pipeline, experiences from previous experiments can be used. Therefore, the quality of each experiment is evaluated and the related quality parameter in the knowledge base is adapted. In this first stage of development, the findings from previous experiments can only be selected from the same type of machine, since it is not exactly known which characteristics influence the quality of results. However, it supports the efficient selection of suitable algorithms and enables the transfer of learning results regarding the same machine types.

Algorithm 1 Cognition module for optimization.

Input: Initial design size s, selection step size θ, algorithm characteristics KB, goal g, available resources r, historical data d_H

```
1  define List for evaluation e
2  define List data d
3  define List pipeline resource consumptions p_r
4  define Parameter x
5  if (|d_H| = 0) then
6      List parameter l ← createInitialDesign(s, KB)
7      forall the (parameter l) do
8          d ← d + applyToCPPS(l)
9          x = l
10 else
11     d, x ← historicalData d_H
12 repeat
13     if (nrIterations % θ = 0 ∨ ζ = 1) then
14         ζ = 0
15         testFunctionSet S ← generateTestFunctions(d)
16         List p ←
           determineFeasiblePipelines(KB, g, d)
17         p ← selectCandidatePipelines(KB, r, p, e)
18         forall the p[i] do in parallel
19             e ← applyPipeline(p[i], S)
20         KB, p_best ← ratePipelines(KB, p, e)
21     x_best ← getBestX(p_best, d, x)
22     if (|x − x_best| ≥ ϵ) then
23         e ← applyToCPPS(x_best)
24         x = x_best
25     d ← d + receiveNewData()
26     if ( since θ/2 steps stagnation ∨ performance
       decrease) then
27         ζ = 1
28 until true;
```

Algorithm 1 provides detailed insights into the cognition module behavior for optimization use cases. At first, needed variables are defined (lines 1–4). If no historical data are available, an initial design representing a list of different parameter configurations is created, the parameters are applied to the CPPS, and the resulting data are stored in the list d.

As several design methods exist and due to the modular design of the architecture, the design method can easily be adapted for special purposes of the CPPS. Currently, a full factorial design is the default configuration, and we recommend space-filling designs. For several design

Table 1 Properties of algorithms; italic properties can be updated by the cognition module

Property	Values
Input data type	Continuous, discrete, hybrid, timed Automata, neuronal net
Output data type	Continuous, discrete, hybrid, timed Automata, neuronal net, time anomaly
Use case class	Optimization (min/max), CM, Anomaly detection, Diagnosis
Algorithm class	Hill-climber, Trajectory, Population, Surrogates
Use multithreads	true, false
Min Training Data	0..n
Prefer usage	true, false
Avoid usage	true, false
Performance	0..1
Computational effort	0..1
RAM usage	0..1

methods and corresponding optimality criteria, we refer to [2, 20, 31].

After the initialization phase, the process of the algorithm selection based on data-driven simulation and benchmarking is started within an infinite loop (line 12). Since we do not want to change pipelines in every iteration, a user-defined step size θ and a variable ζ are checked, to potentially change the pipeline only after each θ step or in situations of prolonged stagnation or performance decrease. Then, a test function set s is generated based on the data d (line 15). Gaussian process simulation is employed for this task [53]. Simulation intends to reproduce the covariance structure of a set of samples (in this case, the process data gathered with the initial design), which maintains the topology of the problem landscape. The intention is to analyze the behavior of the performance for candidate algorithms on problems similar to the CPPS problem. The generation of several different instances, either via spectral simulation or simulation by decomposition (see [53] for details), allows a benchmarking of potentially feasible algorithms. With this benchmark, the algorithms can be analyzed regarding their resource consumption (computation time and memory consumption) and performance (based on their rank, not the achieved values, as the problem instances most likely are different in that regard) even for a larger number of production cycles. We assume that the resource consumption depends on the current machine load, the number of function evaluations to perform, and the dimensionality of the problem, but not on the problem landscape structure. Consequently, the resource consumption can be analyzed quite accurately using simulations.

The algorithm description from the knowledge base is used to determine a list of all feasible pipelines in line 16. Based on the available resources, the knowledge base, and possibly existing earlier evaluations, the most suitable pipelines are selected from the list of feasible pipelines in line 17. This ensures that pipelines can be excluded, which are already known to not compute a result in time. These pipelines are applied in parallel to the test function set s, which is used for benchmarking and possible tuning of the pipelines (line 18–19), where the applied test instances are drawn randomly.

The final decision, which pipeline is used, is described in line 20. As the overall performance measure, a weighted normalized aggregation of the achieved performance on the simulation instance, the memory consumption, and the used CPU time is computed and assigned to each pipeline. For an overview and discussion of issues and best practices of benchmarking in general, and performance assessment in particular, we refer to [3]. A normalized processing time, computed by dividing the used CPU time by the runtime of a standard algorithm, is suggested by Johnson and McGeoch [21] and Weise et al. [48]. Inspired by this idea, we chose to consider the baseline comparator, i.e., the random search, as the reference algorithm. The goal is to reward efficient usage of resources, leading to higher accuracy compared to the baseline and relative to the competing algorithms. On the other hand, algorithms that do not spend too many resources, due to their simplicity, should not be overrated, when they do not achieve proper objective function values. Otherwise they might be good fallback choices, when system load is high and few resources are available, at least if they perform better than the baseline. Consequently, algorithms performing worse than the baseline will be removed from this iteration.

The best pipeline in the remaining list is chosen for application on the CPPS. If the list is empty, the current x will be returned. Should the new x_{best} differ significantly from the current x, the new x_{best} is applied to the CPPS for the next iteration (lines 22–24).

The new data produced by the CPPS are added to d (line 25). If there is no significant improvement after half of the step size, the ζ is set to 1, to perform an unscheduled algorithm selection cycle. This should help at the beginning of the process to recover poor decisions.

4 Implementation and evaluation

This section presents the implementation and integration of the *Cognition* on the BDP. Subsequently, it describes the real-world evaluation use case and the obtained results.

The implementation, which is described in this section, is available on GitHub.[1]

4.1 Big data platform

The BDP runs on Kubernetes, which orchestrates the different services. Kubernetes enables a declarative cluster management, e.g., the user submits a specification of the desired application state to Kubernetes and its controller takes the required measures to reach this state. The user composes the application of different building blocks, which fits the modular approach of the CAAI architecture very well. The smallest building block in Kubernetes is called a pod and consists of one or more application containers and optionally a volume for data storage. The BDP uses two higher level Kubernetes objects for most of the services, namely the deployment and the job.

The deployment is used for all modules of the BDP that require continuous operation, e.g., the cognitive module, the messaging solution, or also the container registry. A deployment, as shown in Fig. 3, specifies the number of pod replicas, which the Kubernetes controller instantiates and monitors on the available nodes in the cluster. The controller will start new instances if a single pod or a complete node fails to get the cluster back to the desired deployment state.

The job is meant for one-off execution of a task, e.g., a part of a data processing pipeline. A job, as seen in Fig. 4, defines a pod and the desired amount of parallelism or the number of allowed retries, if the job fails during execution. The Kubernetes controller will track the job progress and manage the whole life cycle of the job to free up resources after completion.

The complete process to dynamically create a data processing pipeline is depicted in Fig. 5. The *Cognition* decides which algorithms should be tested on the current use case based on information about available cluster resources from the monitoring module and the knowledge on available algorithms and their properties. The cognitive module can specify and submit new jobs to dynamically build data processing pipelines. The controller subsequently pulls the container images for the given jobs from the container registry and instantiates them.

Kubernetes defines all resources declaratively via the YAML Ain't Markup Language (YAML) and Listing YAML 1 shows an exemplary job definition. Lines 1–2 define the type of Kubernetes resource, while lines 3–4 specify job metadata, e.g., assign the algorithm name as an identifier for the job, and line 5 and onward describe the job at hand. Lines 6–8 specify the Kubernetes configuration for the job, i.e., the number of retries (line 6), the maximum allowed time for processing (line 7), and the time before a finished job will be deleted by the controller (line 8). The actual job can consist of one or more containers and the associated configuration is shown in lines 9–15. A job template contains the container image (line 13) and the command to execute within the container on start-up (line 14), e.g., execute the random forest implementation with Python. It is possible for the cognition to adjust the arguments to pass into the container (line 15), e.g., how many trees the random forest algorithm will create or which is the criterion for a split. Thus, the *Cognition* can dynamically configure and instantiate a job that applies the random forest algorithm on the production data.

YAML 1 Kubernetes job definition.

```
1  apiVersion: batch/v1
2  kind: Job
3  metadata:
4      name: Random_Forest
5  spec:
6      backoffLimit: 5
7      activeDeadlineSeconds: 20
8      ttlSecondsAfterFinished: 60
9      template:
10         spec:
11         containers:
12         -name: random_forest
13             image: caai/random_forest
14             command: [”python”, ”-u”, ”random_forest.py”]
15             args: [”NumberOfTrees=5”]
16     restartPolicy: OnFailure
```

The *Cognition* uses the knowledge base to compose feasible pipelines and configure the available algorithms. In this work, we used the YAML format also for the hierarchical knowledge representation as a simple and straightforward way to represent algorithm knowledge that fulfills the requirements for extensibility and modularity. The representation structure is based on the 4-step process to define declarative goals, presented earlier in Fig. 2. Listing YAML 2 shows the exemplary knowledge representation of the random forest algorithm. Lines 1–3 represent the declarative user goals and the following lines describe the algorithms that are suited to achieve the individual goals. Algorithms are described with the parameters, metadata about the algorithm, and the required input. Parameters (lines 6–11) are described by the type, and

[1] https://github.com/janstrohschein/KOARCH/tree/master/Use_Cases/VPS_Popcorn_Production/Kubernetes

minimum, maximum, and default values of the parameter. The metadata (lines 12–20) is a representation of Table 1, for a certain algorithm. Not initialized values are indicated by −1. Those values are determined during the run time of the algorithms through the Cognition. The *efficiency* is calculated based on the model quality and the utilized resources as a selection criterion for the *Cognition*. The input (line 21) for the algorithm is described as a string, e.g., preprocessed data. All algorithms that provide this type of input are candidates for a pipeline and are described on the next hierarchy level. The *Cognition* repeats this selection process until the required input is designated as raw data which marks the end of a pipeline. Thus, the representation enables a fast and easy composition of feasible pipelines and provides all necessary information for the *Cognition* to evaluate those pipelines and decide which is the best suited for the production process.

YAML 2 Knowledge representation.

```
1  Optimization:
2    minimize:
3      Minimum:
4        Algorithms:
5          Random Forest:
6            parameter:
7              NumberOfTrees:
8                type: int
9                default: 4
10               min: 1
11               max: 100
12           metadata:
13             Class: Surrogate
14             Performance: -1
15             Computational effort: -1
16             RAM usage: -1
17             Use multithreads: false
18             Min training data: 5
19             Prefer usage: true
20             Avoid usage: false
21           input: preprocessed data
```

4.2 VPS use case

We use the versatile production system (VPS), which is located in the SmartFactoryOWL, for evaluation of the cognitive module. The VPS is a modular production system, which processes corn to produce popcorn which is used as packaging material. Due to its modularity, it can be adapted to the current order easily. Efficiently operating the VPS is a challenge because many parameters influence the result, which cannot be measured inline, e.g., the moisture of the corn. More details about the use case can be found in [15]. Thus, a data-driven optimization is a promising method to increase efficiency, which is performed using the CAAI and the introduced cognitive module.

The goal is to operate the VPS at or near the optimum. This is difficult because the quality of the corn has a large influence on the optimal process parameter. Depending on the quality, the change of volume between the corn and the popcorn differs in a large range and can not be measured beforehand. Thus, an optimization during the plants' run time is necessary.

In this paper, we take a typical application, where the corn is delivered in batches. Since the quality between batches can differ, optimization is performed for each batch. The batches are optimized independently, so parameters of the production plant, such as the size of a portion, can be changed between batches. This describes the typical usage of the plant.

The amount of corn that filled the reactor has to be optimized to get the required amount of popcorn. The overage of popcorn produced in one batch, or not fully filled boxes, cannot be used, so it is wasted. The optimum is a trade-off between three minimization functions: the energy consumption (f_1), the processing time (f_2), and the amount of corn needed for a small box (f_3). These functions are conflicting to some degree. The optimization result is a parameter value x for the dosing unit that indicates the run time of the conveyer, and thus influences the amount of corn. As the given optimization problem can be regarded as relatively straightforward, we will apply a single objective optimization algorithm and compute a weighted sum of the objectives. This results in the following optimization problem:

$$\min \sum_{i=1}^{3} w_i f_i(x); \quad \text{w.r.t. } w_i > 0 \text{ and } \sum_{i=1}^{3} w_i = 1 \tag{1}$$

The scalar weights of the corresponding objectives, w_i, are chosen based on user's preferences. As a default, equal weights are used.

4.3 Results

To evaluate the cognitive module, data from the real-world VPS was acquired. A set of 12 different settings for the run time of the conveyer was used, each repeated three times which results in 36 production cycles in total. This data was used to fit a Gaussian process model. To retrieve an accurate simulation of the popcorn production, a conditional simulation was used (hereafter referred to as ground truth), so consequently, the model respects the data points, whereas the test instances, which will be used to perform the

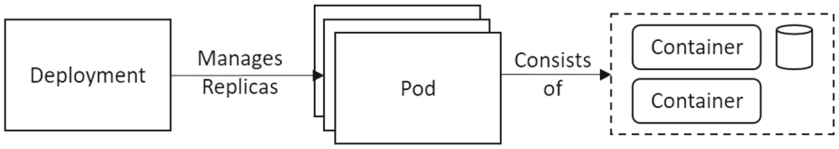

Fig. 3 A deployment defines the life cycle of a pod, which consists of one or more containers and an optional volume for information storage. Submitting a deployment to the controller instructs Kubernetes to instantiate the pod on a node and ensure its continuous availability in the cluster

benchmark experiments, will use unconditional simulation (hereafter referred to as simulations).

The resulting problem instances are shown in Fig. 6. It can be seen that the simulation instances (shown as solid blue lines) arguably reflect the character of the ground truth (the dashed line), but clearly have different forms of peaks and valleys.

R version 3.6.3 was the used software platform [36] to perform the experiments. To estimate the consumption of CPU time per optimizer, the *tictoc* package in version 1.0, and, for the monitoring of the memory consumption, the *profmem* package in version 0.5.0 are used. The Gaussian process simulations for continuous problem spaces are computed using the *COBBS* package in version 1.0.0, available on GitHub [2]. We use the differential evolution implementation in the *DEoptim* package in version 2.2-5, the generalized simulated annealing (Generalized SA) implementation in the *GenSA* package in version 1.1.7, the Kriging and random forest-based optimization from the *SPOT* package version 2.0.6, and the L-BFGS-B implementation from the *stats* package included in the R version. The considered algorithms with chosen parameters and corresponding values are summarized in Table 2. As a detailed description of the algorithms is out of scope of this paper, we refer to related publications. Uniform random sampling does not belong to any of the before mentioned algorithm families, but is added as a baseline comparator and works without any control parameter. The L-BFGS-B, a limited memory variant respecting bound constraints of the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) algorithm, is controlled by the parameter l mm, which sets the number of BFGS updates [9]. The differential evolution algorithm is mainly controlled by the number of individuals per generation (popsize), the evolution strategy applied (strategy), the stepsize (F), the crossover probability (CR), and the crossover speed adaptation (c); for details, we refer to [32]. Generalized SA is a variant of the famous simulated annealing global optimization algorithm, mimicking the cool-down process in metallurgy [50]. The parameter temp initializes the starting temperature, q_v and q_a are parameters for the visiting and acceptance distribution, respectively, controlling the candidate solutions, and the acceptance of candidates with worse performance values to potentially escape local optima. Details about SPOT can be found in [4].

Please note that some values were chosen according to the relatively low budget of function evaluations, which is suitable for the given use case. This is especially the case for parameters that, e.g., control the number of candidate solutions per iteration. See, for example, the designSize, which is the size of the initial design of both chosen surrogate-model–based optimizers, or the popsize, which is the number of individual solutions for each iteration of the differential evolution algorithm.

At first, the applicability of the simulations for the VPS problem is analyzed experimentally. We assume a correlation between the performance of the available algorithms on the ground truth and the simulations.

Figures 7 and 8 show the rank of the performance of the algorithms on the corresponding problem instance, i.e., the ground truth or the simulation. Even if some algorithms change their rank over time, a quite strong correlation of the difficulty between the instances can be seen. A pairwise correlation analysis of the algorithm performance on the two different types of objectives, either the ground truth or the simulations, using the Pearson method is applied. The results are shown in Table 3 and reveal a both high and significant correlation between the performance of the algorithms on the different objective functions with a correlation coefficient of 0.823 and a low p-value (2.2e−16). This supports our assumption that the usage of simulations to benchmark algorithms is valid.

Figure 9 shows the performance w. r. t. the best-achieved objective function value of the implemented algorithms on the ground-truth objective as a mean result over ten repetitions. GenSA and L-BFGS-B perform relatively inconsistently, and are not always able to beat the random search. For GenSA, this might be due to the modality of the landscape and the chosen parameters, which are the default settings. L-BFGS-B seems to struggle with the modality, and can be most beneficial on unimodal problems. Nevertheless, it is worth to see if it can compete, or not. Differential evolution is increasing its performance over time, and can expectably be a consistently performing

[2] https://github.com/martinzaefferer/COBBS

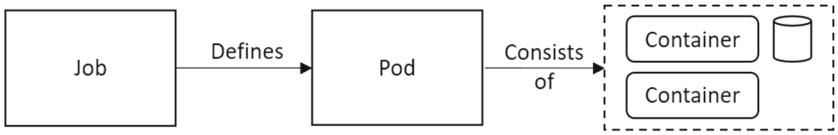

Fig. 4 A job defines a pod, which consists of one or more containers and an optional volume for information storage. The job specifies a workload to process with a desired number of completions or parallelism, to run the same algorithm multiple times, or launch several pods to work on one job in parallel

competitor in further iterations. It is to note that only both surrogate-based optimizers perform consistently better than the baseline comparator, the simple random search. Therefore, only optimizers achieving better objective function values than the baseline will be considered for further algorithm selection.

To get the an overview of the full picture next, we look at the resource consumption of the implemented algorithms.

Figure 10 depicts the increase in the mean of the CPU usage of the algorithms on the ground-truth objective over ten repetitions each. Please be aware that the shown CPU consumption includes the time to evaluate the objective function, which basically performs a Gaussian process prediction and related matrix operations. This effect is nearly identical for each algorithm, but will surely further increase over time. The CPU consumption of the random search can be seen as a baseline to estimate the pure CPU consumption of the objective function evaluation.

The development of the memory consumption on the ground truth is shown in Fig. 11 for each algorithm as a mean value over ten repetitions. It can be seen that some algorithms do not store much data during the process of the optimization, whereas the surrogate-based optimizers show a linear increase in memory consumption. This reveals an important issue, when the memory consumption is integrated into the rating of the algorithms: it can overrate poor performers, w. r. t. to the achieved objective function value.

To conclude the findings, we can summarize that the best performing algorithms for this setup also consume significantly higher amounts of CPU time and memory. This needs consideration in the algorithm selection process even if system resources are scarce and weak optimizers appear beneficial due to their low demands on computational power and available memory. Consequently, optimizers performing worse than the baseline will be excluded from the selection process, as the baseline itself is a cheap fallback.

A multi-objective performance assessment of algorithms is highly encouraged since consumptions of available resources and solution quality mutually influence each other [6]. Consequently, our goal is to integrate CPU time and memory consumption into the overall performance rating. Please note that resource consumptions are dependent on system load, hardware, and programming language. Therefore, results will not be easily reproducible nor generalizable, but will reflect real-time circumstances.

The overall performance on the simulations will be evaluated as follows. For each budget, the random search performances according to the achieved objective function value, the consumed memory, and the processed CPU time will be taken as a reference for the competitors. Achieved

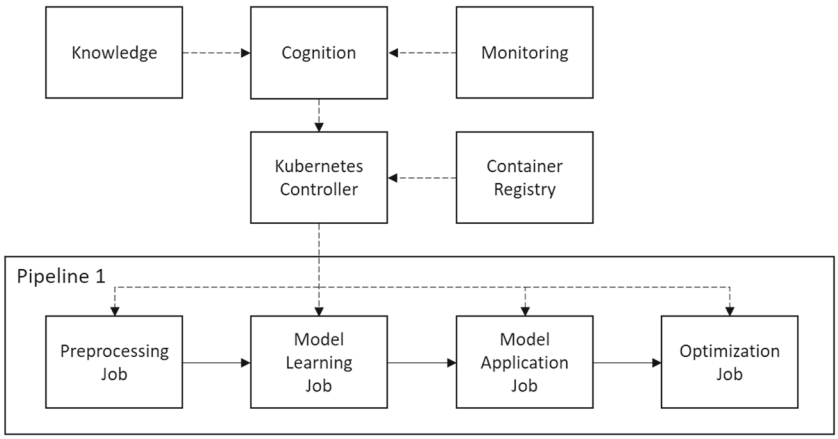

Fig. 5 The *Cognition* creates new data processing pipelines based on information about the available algorithms from the knowledge module and information on current resource usage provided by the monitoring module. The *Cognition* decides on one or more pipelines and instructs the Kubernetes Controller to instantiate the data processing modules for each. The Kubernetes Controller loads the container images from the registry and starts all the jobs that form the data processing pipeline

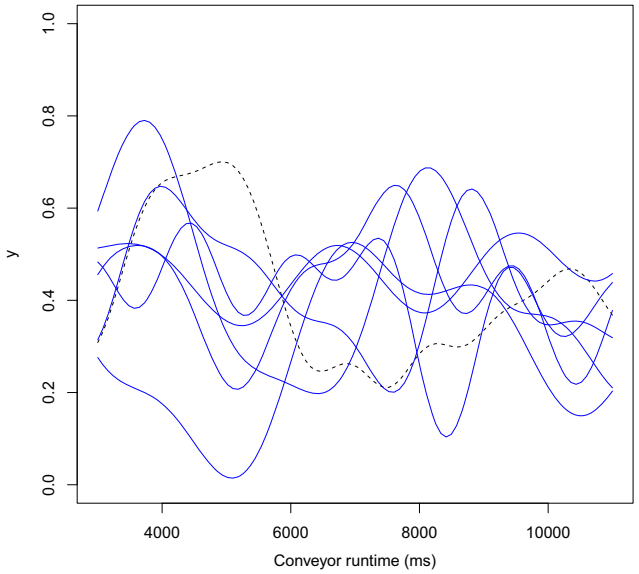

Fig. 6 This plot shows the resulting VPS problem instances based on the real-world data taken from the machine. The dashed line shows the conditional simulation, and the solid blue curves represent the unconditional simulations used as test instances for further benchmarking of the algorithm pipelines. The x-axis shows the run time of the conveyor in ms, and the y-axis shows the equally weighted normalized aggregated objective function value of the objectives' process time, amount of corn, and energy consumption

Table 2 Settings of parameter ranges and corresponding default values of chosen optimizers

Parameter	Range	Default	family
Differential evolution			Population
popsize	$\mathbb{N}_+$	5	
strategy	{1, 2, 3, 4, 5}	2	
F	[0, 2]	0.8	
CR	[0, 1]	0.5	
c	[0, 1]	0.5	
Generalized SA			Trajectory
temp	$\mathbb{N}_+$	100	
q_v	$\mathbb{R}$	2.5	
q_a	$\mathbb{R}$	-1	
Kriging (SPOT)			Surrogate
designSize	$\mathbb{N}_+$	7	
designType	$\{Lhd, Uniform\}$	Lhd	
Random forest (SPOT)			Surrogate
nrTrees	$\mathbb{N}_+$	500	
designSize	$\mathbb{N}_+$	7	
designType	$\{Lhd, Uniform\}$	Lhd	
L-BFGS-B			Hill-climber
lmm	$\mathbb{N}_+$	5	
Uniform random sampling			Baseline

objective function values will be computed as relative improvement compared to the baseline value. If there is no improvement, the algorithm will be removed. The memory and CPU consumption of each remaining algorithm will be divided by the baseline reference values. Each of the factors will be normalized, such that the best performing optimizer gets the value 1 and the worst the value 0 assigned, while the rest are scaled in between. Multiplying each normalized factor with the factor-weight calculates an aggregated performance value. We generally recommend to set a high weight for the optimization quality, and minor weights to memory and CPU time (e.g., $0.8 \times objective$, $0.1 \times memory$, $0.1 \times CPU$). This may be adapted, if the system load is very high.

We summarize two scenarios, i.e., two different weighing vectors. The first with an high focus on the objective function value (0.8), and both resource measures set to 0.1. The second with 0.5 assigned to the objective, and relatively high values on both resource measures (0.25 each).

The results show that the surrogate-based optimizers are in both scenarios, i.e., with different weights applied to aggregate measures on the simulation instances, valid choices (see Fig. 12). Please note that only algorithms superior to random search are ranked for each production cycle. As random forest is somewhat more economical compared to Kriging, it is rated the highest rank for both cases. When the focus on the resources is higher (the second scenario), it can be seen that algorithms like differential evolution or GenSA can be ranked equally to Kriging, even if they achieve significantly worse objective function values. It can be seen that the number of considered algorithms differs over time, as not all are superior to random search all the time. This results in missing entries in the ranking.

As algorithm tuning can lead to performance increases of the applied algorithms, we exemplified the usage of the test instances to tune differential evolution for the VPS use case. The procedure was as follows. For two different budgets of 20 and 30 function evaluations, a test instance was randomly selected to perform tuning using SPOT. The performance of Differential evolution was evaluated before and after tuning with 10 repetitions each, using the default parameters for the former, and the best found parameter values during tuning for the latter case. SPOT was set up with a budget of 30 evaluations. The results are shown in Fig. 13. It can be seen that the algorithm performance variance has clearly decreased; and thus, the differential evolution can perform more robust after tuning. The other algorithms can be tuned similarly, but the tuning potential is considerably lower. Tests showed no significant difference in their performance after tuning, so we left them out. Selecting and implementing an efficient tuning strategy, embedded in the online algorithm selection procedure, is a challenging task and therefore not focus of this paper. A follow-up study will

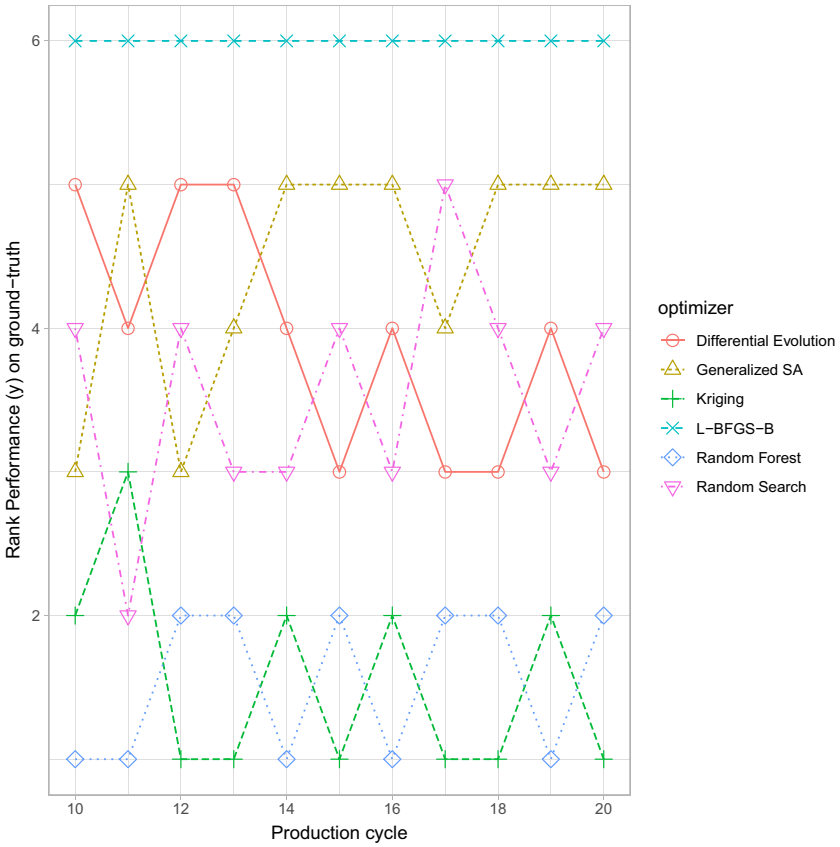

Fig. 7 This plot shows the rank-based performance of the algorithms on the ground truth over the number of production cycles. Lower ranks represent better performance

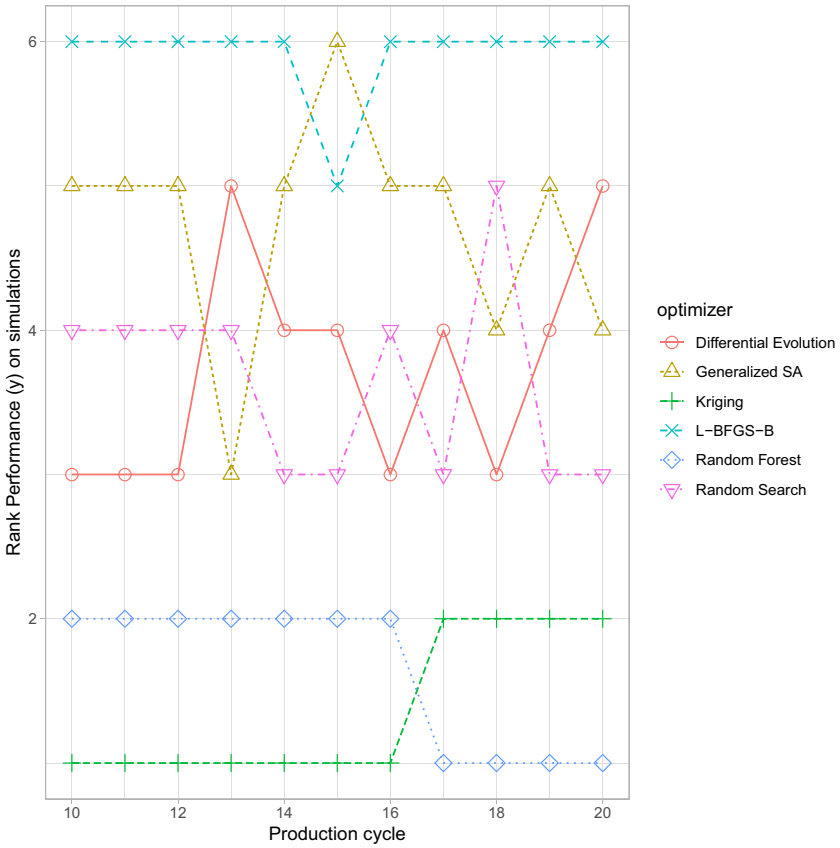

Fig. 8 This plot shows the rank-based performance of the algorithms on the simulations over the number of production cycles. Lower ranks represent better performance

Table 3 Results of the Pearson's correlation between the algorithms performance on the ground truth and on the simulations

Correlation coefficient	0.823
95% confidence interval	[0.725, 0.888]
t	11.575
p-value	2.2e−16
df	64

analyze the questions, which algorithms should be tuned and when, and with how much computational effort, to find the best tuning strategy.

Please note that the Gaussian process–based simulation currently also comes with some limitations. The model fit gets more and more time consuming with the increase in the amount of data or the dimensionality of the problems search space, i.e., the number of parameters x. In situations where the simulations can not be performed efficiently anymore, we recommend a switch to different simulation models.

Please note that the experiments do not have the purpose to demonstrate superiority or higher usability of any of the algorithms over others. The intention is to have some tools available for a kind of unknown problem and decide by some experiments, which might most likely be the best to use in the moment, w. r. t. the currently available resources.

5 Summary and outlook

In this paper, we introduce a cognitive module that automatically selects and tunes pipelines for optimization in CPPS based on the current environment. It continues ongoing work on the CAAI architecture.

We define a four-step process, i.e., selecting the use case, the relevant signals, the feature, and the use case goal, to define declarative goals for the *Cognition*. In combination with the parameters to control and adapt the CPPS, which are defined in the *Knowledge*, an optimization problem can be formulated. The *Cognition* uses this information to retrieve feasible pipelines from the knowledge base. Data-driven simulations enable the evaluation of the retrieved algorithms and allow to select the best candidates based on performance metrics and provided user preferences, e.g., prediction quality or resource usage. The selected algorithms use the process data to generate and evaluate models and optimization algorithms during production to find the best solution and continuously adjust the production parameters. Implementing online learning is a distinct advantage in contrast to AutoML methods, which collect batches of data for analysis after the production concluded. The *Cognition* uses the pipeline results to adjust the knowledge base and improves the algorithm selection for a given use case over time. Instantiating additional algorithms

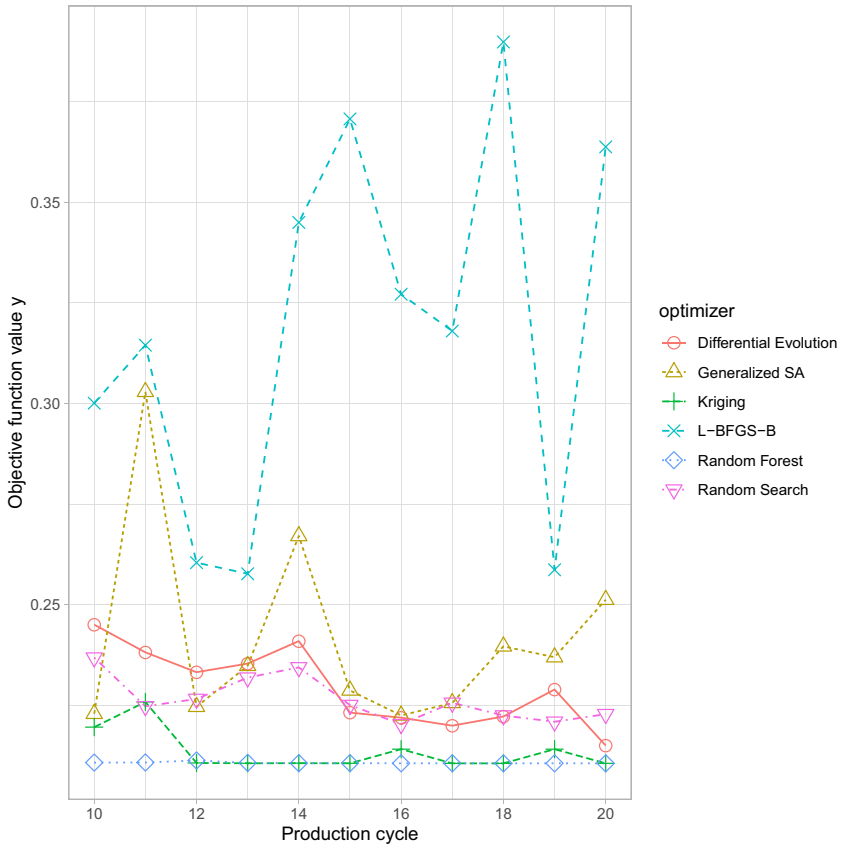

Fig. 9 This plot shows the best achieved objective function values of each algorithm over the budget. The values show the mean performance over 10 repetitions for each algorithm and budget on the ground truth. Lower objective function values are better

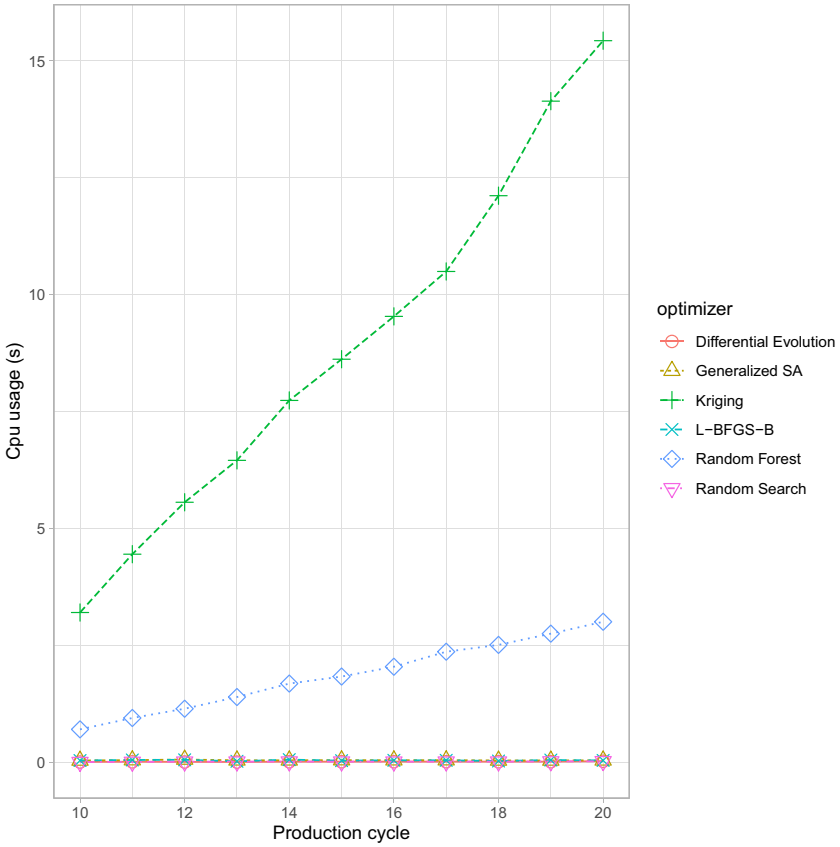

Fig. 10 This plot shows the used CPU time of the applied algorithms over the budget. The values show the mean CPU time over 10 repetitions for each algorithm and budget on the ground truth

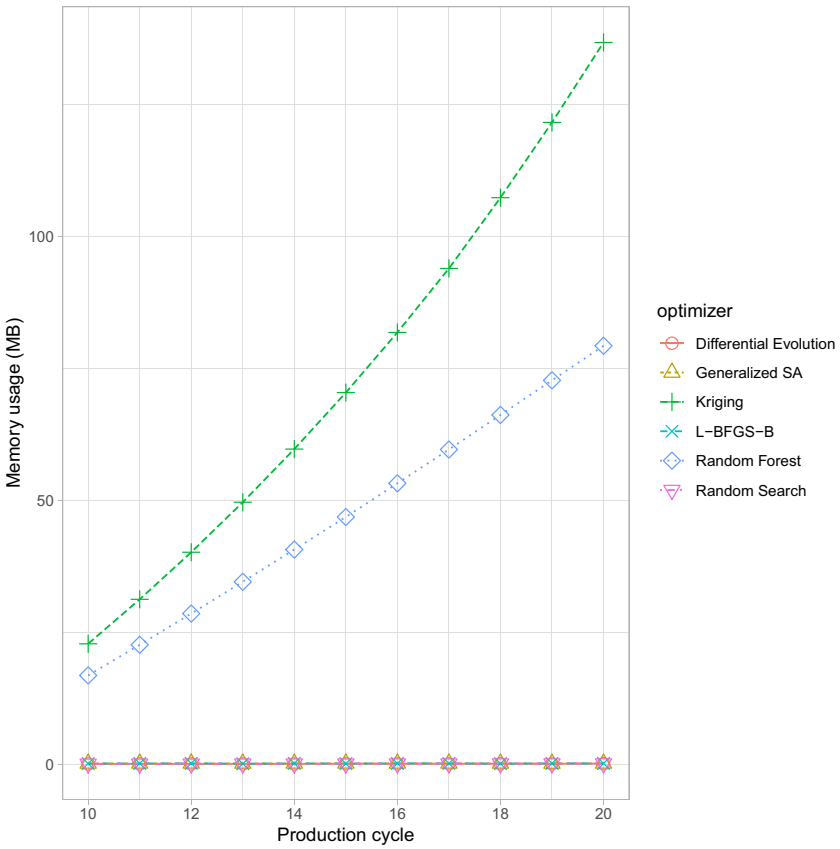

Fig. 11 This plot shows the memory consumption of the applied algorithms over the budget. The values show the mean memory usage over 10 repetitions for each algorithm and budget on the ground truth

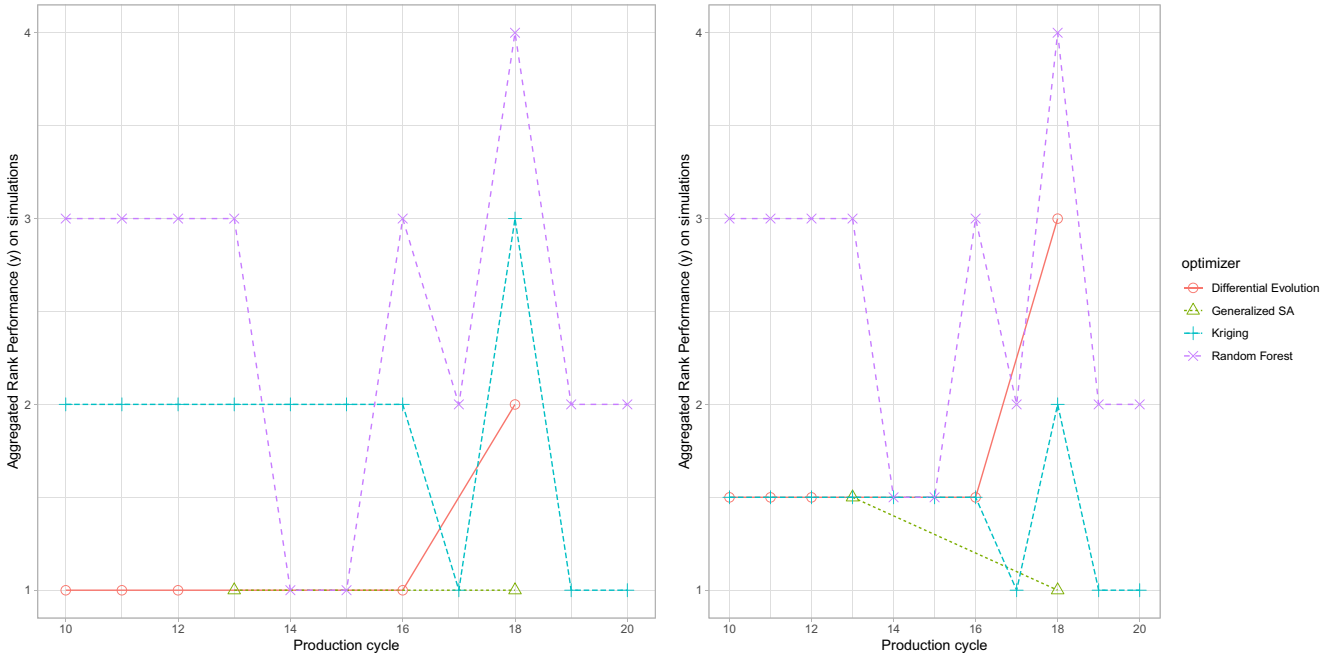

Fig. 12 This plot shows the resulting rank of the aggregated quality measure over the production cycles where a higher rank indicates a better result. Only algorithms superior to random search are ranked for each production cycle. The left side shows the first scenario, with weights focussing on the optimization objective value, i.e., $0.8 \times objective$, $0.1 \times memory$, $0.1 \times CPU$). The right side displays the second scenario with the following weights: $0.5 \times objective$, $0.25 \times memory$, $0.25 \times CPU$). This reflects situations, where resources should be consumed more economically

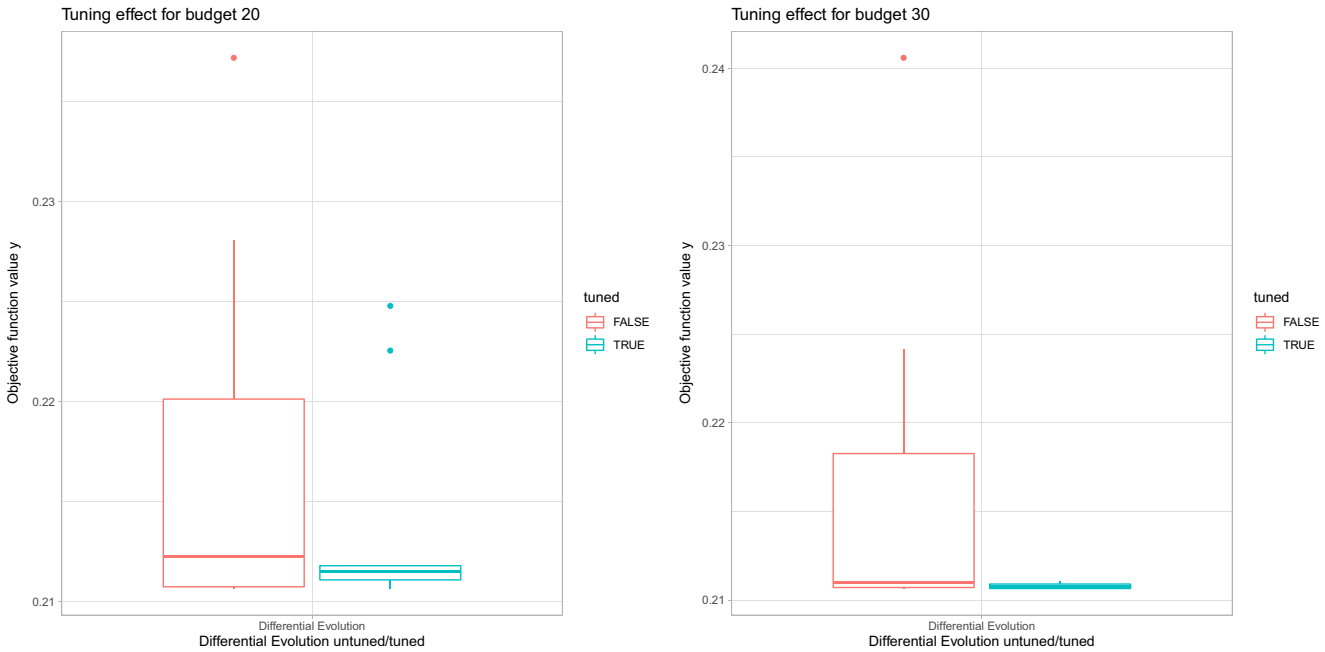

Fig. 13 This plot shows the effect of parameter tuning of the differential evolution algorithm. The left side shows the effect for a budget of the 20 evaluations on the VPS problem, and the right side shows the results for a budget of 30 evaluations on the VPS problem. Tuning was performed on randomly drawn test instances. The algorithm has become more robust in both scenarios, as the performance variance decreased clearly

during production and learning from the results is possible due to the implementation of the CAAI architecture on Kubernetes. We present the available Kubernetes resources and the process to dynamically deploy workloads. The *Cognition* is able to evaluate the system performance and schedule additional machine learning pipelines through the Kubernetes scheduler if computational resources are available. The approach is evaluated on a real-world use case and demonstrates how the CAAI architecture uses the *Cognition* to reduce the manual implementation effort for AI algorithms in CPPS.

The evaluation demonstrates how CAAI uses the cognitive module to automatically select and tune algorithms to perform online optimization for the popcorn production in the VPS. This is possible without the time-consuming acquisition of training data through the operators and does not require data science knowledge regarding the algorithms. Depending on the difficulty and complexity of the problem at hand, the data-driven simulation of the real-world problem allows an efficient benchmark of the available algorithms and enables the proper selection of the most suitable one. The implementation of CAAI on Kubernetes allows to dynamically instantiate the selected algorithms in machine learning pipelines during production.

Several challenges are open for future work. The presented use case is a real-world production plant with a rather straightforward optimization problem—nevertheless, the evaluation is valuable and difficult for the operator to perform manually. Although we focus on the optimization use case, CAAI is designed to solve a diverse set of use cases, such as condition monitoring, predictive maintenance, or diagnosis. Therefore, the set of algorithms and the cognitive component have to be extended to address other use cases as well.

Furthermore, the cognitive component can be improved, e.g., by considering the total number of products in a batch, realizing transfer learning to reduce the run-up time after a change, recognizing changes in the input material and adapting the parameters for the model learning, or by an automated feature selection. The behavior of the *Cognition* on more complex optimization problems still needs to be evaluated. Therefore, an adjusted algorithm portfolio, e.g., adding algorithms like PSO or covariance matrix adaptation evolution strategy (CMA-ES), can be considered.

Additionally, the most efficient tuning strategy for the online algorithm selection process should be researched. The challenge will be to detect the algorithms with the highest tuning potential and implement an efficient tuning schedule for the overall process, as tuning can quickly become computationally expensive. It can be considered to avoid tuning of algorithms which finally will not be the best candidates, e.g., according to a selection likelihood. Consequently, predicting the potential of tuning for considered algorithms and investing the computational resources efficiently are goals for future work.

Funding Open Access funding enabled and organized by Projekt DEAL. The work was supported by the German Federal Ministry of Education and Research (BMBF) under the project "KOARCH" (funding code: 13FH007IA6 and 13FH007IB6).

Data availability The data and materials can be found in the corresponding GitHub repository: https://github.com/janstrohschein/KOARCH/tree/master/Use_Cases/VPS_Popcorn_Production/Kubernetes/experiments

Code availability The implementation can be found in the corresponding public GitHub repository: https://github.com/janstrohschein/KOARCH/tree/master/Use_Cases/VPS_Popcorn_Production/Kubernetes

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Altintas I, Marcus K, Nealey I, Sellars SL, Graham J, Mishin D, Polizzi J, Crawl D, Defanti T, Smarr L (2019) Workflow-driven distributed machine learning in CHASE-CI: A cognitive hardware and software ecosystem community infrastructure. In: Proceedings - 2019 IEEE 33rd international parallel and distributed processing symposium workshops (IPDPSW), vol 2019, pp 865–873. https://doi.org/10.1109/IPDPSW.2019.00142
2. Atkinson AC, Fedorov VV, Herzberg AM, Zhang R (2014) Elemental information matrices and optimal experimental design for generalized regression models. J Stat Plan Inference 144:81–91. 10.1016/j.jspi.2012.09.012, http://www.sciencedirect.com/science/article/pii/S0378375812003060. International Conference on Design of Experiments
3. Bartz-Beielstein T, Doerr C, Bossek J, Chandrasekaran S, Eftimov T, Fischbach A, Kerschke P, Lopez-Ibanez M, Malan KM, Moore JH, Naujoks B, Orzechowski P, Volz V, Wagner M, Weise T (2020) Benchmarking in optimization: best practice and open issues. arXiv:2007.03488
4. Bartz-Beielstein T, Gentile L, Zaefferer M (2017) In a nutshell: sequential parameter optimization. arXiv:1712.04076
5. Bhatti MA (2000) Optimization problem formulation. Springer New York, New York, pp 1–45. https://doi.org/10.1007/978-1-4612-0501-2_1

6. Bossek J, Kerschke P, Trautmann H (2020) A multi-objective perspective on performance assessment and automated selection of single-objective optimization algorithms. Appl Soft Comput 88:105901. https://doi.org/10.1016/j.asoc.2019.105901. http://www.sciencedirect.com/science/article/pii/S1568494619306829
7. Bunte A, Fischbach A, Strohschein J, Bartz-Beielstein T, Faeskorn-Woyke H, Niggemann O (2019) Evaluation of cognitive architectures for cyber-physical production systems. In: 24nd IEEE international conference on emerging technologies and factory automation (ETFA). Zaragoza, Spain
8. Burns B, Beda J, Hightower K (2019) Kubernetes: up and running, O'Reilly, Newton
9. Byrd RH, Lu P, Nocedal J, Zhu C (1995) A limited memory algorithm for bound constrained optimization. SIAM J Sci Comput 16(5):1190–1208
10. Casquero O, Armentia A, Sarachaga I, Pérez F, Orive D, Marcos M (2019) Distributed scheduling in Kubernetes based on mas for fog-in-the-loop applications. In: 2019 24th IEEE international conference on emerging technologies and factory automation (ETFA). pp 1213–1217
11. Eiben A, Smit S (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol Comput 1(1):19–31. https://doi.org/10.1016/j.swevo.2011.02.001, https://www.sciencedirect.com/science/article/pii/S2210650211000022
12. Feurer M, Eggensperger K, Falkner S, Lindauer M, Hutter F (2020) Auto-sklearn 2.0: the next generation. arXiv:2007.04074
13. Feurer M, Klein A, Eggensperger K, Springenberg JT, Blum M, Hutter F (2015) Efficient and robust automated machine learning. In: Proceedings of the 28th international conference on neural information processing systems - volume 2. MIT Press, Cambridge. https://doi.org/10.5555/2969442.2969547
14. Fischbach A, Bartz-Beielstein T (2020) Improving the reliability of test functions generators. Appl Soft Comput 92:106315. https://doi.org/10.1016/j.asoc.2020.106315, http://www.sciencedirect.com/science/article/pii/S1568494620302556
15. Fischbach A, Strohschein J, Bunte A, Stork J, Faeskorn-Woyke H, Moriz N, Bartz-Beielstein T (2020) CAAI—a cognitive architecture to introduce artificial intelligence in cyber-physical production systems. Int J Adv Manuf Technol 111(1):609–626. https://doi.org/10.1007/s00170-020-06094-z
16. Fischbach A, Zaefferer M, Stork J, Friese M, Bartz-Beielstein T (2016) From real world data to test functions. In: Hoffmann F, Hüllermeier E (eds) Proceedings. 26. Workshop Computational Intelligence. KIT Scientific Publishing, Dortmund, pp 159–177
17. Fusi N, Sheth R, Elibol M (2018) Probabilistic matrix factorization for automated machine learning. In: Proceedings of the 32nd international conference on neural information processing systems. Curran Associates Inc., Red Hook, pp 3352–3361. https://doi.org/10.5555/3327144.3327254
18. Glover F (1989) Tabu search—part i. ORSA J Comput 1(3):190–206
19. Hu L, Xie N, Kuang Z, Zhao K (2012) Review of cyber-physical system architecture. In: Proceedings - 2012 15th IEEE international symposium on object/component/service-oriented real-time distributed computing workshops, ISORCW 2012, pp 25–30. https://doi.org/10.1109/ISORCW.2012.15
20. Husslage BG, Rennen G, Van Dam ER, Den Hertog D (2011) Space-filling Latin hypercube designs for computer experiments. Optim Eng 12(4):611–630
21. Johnson DS, McGeoch LA (2007) Experimental analysis of heuristics for the STSP. Springer US, Boston, pp 369–443. https://doi.org/10.1007/0-306-48213-4_9
22. Kagermann H, Wahlster W, Helbig J (2013) Securing the future of German manufacturing industry Recommendations for implementing the strategic initiative NDUSTRIE 4.0. acatech – National Academy of Science and Engineering, Berlin
23. Karkalos N, Efkolidis N, Kyratsis P, Markopoulos A (2019) A comparative study between regression and neural networks for modeling al6082-t6 alloy drilling. Machines 7(1):13. https://doi.org/10.3390/machines7010013
24. Kaur K, Garg S, Kaddoum G, Ahmed SH, Atiquzzaman M (2019) Keids: Kubernetes based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem. IEEE Internet Things J:1–1
25. Kerschke P, Trautmann H (2019) Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evol Comput 27(1):99–127. https://doi.org/10.1162/evco_a_00236
26. Kirkpatrick S, Gelatt CD, Vecchi MP et al (1983) Optimization by simulated annealing. Science 220(4598):671–680
27. Škvorc U, Eftimov T, Korošec P (2020) Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. Appl Soft Comput 90:106–138. https://doi.org/10.1016/j.asoc.2020.106138, http://www.sciencedirect.com/science/article/pii/S1568494620300788
28. Liu DC, Nocedal J (1989) On the limited memory bfgs method for large scale optimization. Math Programm 45(1-3):503–528
29. Mersmann O, Bischl B, Trautmann H, Preuss M, Weihs C, Rudolph G (2011) Exploratory landscape analysis. https://doi.org/10.1145/2001576.2001690. http://dl.acm.org/ft_gateway.cfm?id=2001690&type=pdf
30. Minsky M (1974) A framework for representing knowledge. Tech. rep., Massachusetts Institute of Technology (MIT), Cambridge, MA USA
31. Morris MD, Mitchell TJ (1995) Exploratory designs for computational experiments. J Stat Plan Inference 43(3):381–402. https://doi.org/10.1016/0378-3758(94)00035-T, http://www.sciencedirect.com/science/article/pii/037837589400035T
32. Mullen K, Ardia D, Gil D, Windover D, Cline J (2011) Deoptim: an r package for global optimization by differential evolution. J Stat Softw, Articles 40(6):1–26. https://doi.org/10.18637/jss.v040.i06 , https://www.jstatsoft.org/v040/i06
33. Muñoz MA, Sun Y, Kirley M, Halgamuge SK (2015) Algorithm selection for black-box continuous optimization problems: a survey on methods and challenges. Inf Sci 317:224–245. https://doi.org/10.1016/j.ins.2015.05.010. http://www.sciencedirect.com/science/article/pii/S0020025515003680
34. Olson RS, Urbanowicz RJ, Andrews PC, Lavender NA, Kidd LC, Moore JH (2016) Automating biomedical data science through tree-based pipeline optimization. In: Squillero G, Burelli P (eds) Applications of evolutionary computation. Springer International Publishing, Cham, pp 123–137. https://doi.org/10.1007/978-3-319-31204-0_9
35. Peng Y, Bao Y, Chen Y, Wu C, Guo C (2018) Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the 13th EuroSys conference eurosys 2018 2018-Janua. https://doi.org/10.1145/3190508.3190517
36. R Core Team (2020) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna. https://www.R-project.org/. Accessed: 2020-11-25
37. Radhakrishnan R, Terpstra J, Tondee M, Kearney A (2019) Machine learning using dell EMC OpenShift container platform. Dell EMC Solutions, USA. https://www.delltechnologies.com/en-us/collaterals/unauth/white-papers/solutions/h17870_kubeflow_machine_learning_wp.pdf . Accessed: 2020-11-25
38. Renau Q, Doerr C, Dreo J, Doerr B (2020) Exploratory landscape analysis is strongly sensitive to the sampling strategy. In: International conference on parallel problem solving from nature. Springer, pp 139–153

39. Rice JR (1976) The algorithm selection problem. Elsevier, Amsterdam, pp 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3, http://www.sciencedirect.com/science/article/pii/S0065245808605203
40. Sibalija T, Petronic S, Milovanovic D (2019) Experimental optimization of nimonic 263 laser cutting using a particle swarm approach. Metals 9(11). https://doi.org/10.3390/met9111147. https://www.mdpi.com/2075-4701/9/11/1147
41. Sibalija TV (2019) Particle swarm optimisation in designing parameters of manufacturing processes: a review (2008–2018). Appl Soft Comput 84:105743. https://doi.org/10.1016/j.asoc.2019.105743, https://www.sciencedirect.com/science/article/pii/S1568494619305241
42. Smith-Miles KA (2009) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput Surv 41:1. https://doi.org/10.1145/1456650.1456656
43. Stork J, Eiben AE, Bartz-Beielstein T (2020) A new taxonomy of global optimization algorithms. Natural Comput. https://doi.org/10.1007/s11047-020-09820-4
44. Subramaniam B, Nielsen N, Doyle C, Deshpande A, Knight J, Leishman S (2018) Abstractions for containerized machine learning workloads in the cloud. In: SysML 2018. http://www.sysml.cc/doc/135.pdf
45. Thornton C, Hutter F, Hoos HH, Leyton-Brown K (2013) Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. Association for Computing Machinery, New York, pp 847-855. https://doi.org/10.1145/2487575.2487629
46. Toussaint J, Cheng K (2002) Design agility and manufacturing responsiveness on the Web. Integr Manuf Syst 13(5):328–339. https://doi.org/10.1108/09576060210429784
47. Verma T, Tiwana APS, Reddy CC, Arora V, Devanand P (2016) Data analysis to generate models based on neural network and regression for solar power generation forecasting. In: 2016 7th international conference on intelligent systems, modelling and simulation (ISMS). pp 97–100. https://doi.org/10.1109/ISMS.2016.65
48. Weise T, Chiong R, Lassig J, Tang K, Tsutsui S, Chen W, Michalewicz Z, Yao X (2014) Benchmarking optimization algorithms: an open source framework for the traveling salesman problem. IEEE Comput Intell Mag 9(3):40–52. https://doi.org/10.1109/MCI.2014.2326101
49. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893
50. Xiang Y, Gubian S, Suomela B, Hoeng J (2013) Generalized simulated annealing for global optimization: the GenSA package. R J 5(1):13–28. https://doi.org/10.32614/RJ-2013-002
51. Yang X, Denis L, Tupin F, Yang W (2019) Sar image despeckling using pre-trained convolutional neural network models. In: 2019 Joint urban remote sensing event (JURSE). pp. 1–4. https://doi.org/10.1109/JURSE.2019.8809023
52. Zaefferer M, Fischbach A, Naujoks B, Bartz-Beielstein T (2017) Simulation based test functions for optimization algorithms. In: Proceedings of the genetic and evolutionary computation conference 201. ACM, Berlin, p 8. https://doi.org/10.1145/3071178.3071190
53. Zaefferer M, Rehbach F (2020) Continuous optimization benchmarks by simulation. In: Bäck T, Preuss M, Deutz A, Wang H, Doerr C, Emmerich M, Trautmann H (eds) Parallel problem solving from nature – PPSN XVI. Springer International Publishing, Cham, pp 273-286