Computing, Telecommunication and Control, 2025, Vol. 18, No. 2, Pp. 120–136.
Информатика, телекоммуникации и управление. 2025. Том 18, № 2. С. 120–136.

Software and Hardware of Computer, Network, Telecommunication, Control, and Measurement Systems

Компьютерные сети, вычислительные, телекоммуникационные, управляющие и измерительные систем

Research article
DOI: https://doi.org/10.18721/JCSTCS.18210
UDC 004.896

A SOFTWARE SYSTEM FOR SURROGATE-BASED PROTOTYPING OF GAS TURBINE BLADES USING SERVERLESS CONTAINERS IN THE CLOUD

G.A. Zhemelev ✉, *P.D. Drobintsev*

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ wws.dev@gmail.com

Abstract. Design optimization of gas turbine blades is a complex multidisciplinary task requiring computationally expensive physics simulations. To perform them, a multitude of computer-aided engineering tools are used, often with machine-learning surrogates for rapid prototyping, all integrated into the optimization cycle. However, current approaches to such integration are hindered by the need for labor-intensive manual setups, vendor lock-in and a lack of scalable, automated workflows. We present a novel cloud-based architecture for building flexible optimization pipelines using containerized components. The proposed solution employs serverless containers, asynchronous messaging and cloud services to ensure the system's scalability, portability and resilience. Additionally, it follows MLOps principles to achieve reproducibility and efficient lifecycle management of machine learning models used in the optimization process. Unlike existing frameworks, our solution minimizes user setup complexity, allows easy integration of various software into the optimization cycle, and avoids vendor lock-in through open-source technologies and standard cloud APIs. Experiments with aerodynamic design optimization of gas turbine blades demonstrate the system's scalability, fault tolerance and successful integration of surrogate models for rapid blades prototyping. Furthermore, the system's flexibility and extensible architecture make it applicable to a broader range of engineering design optimization tasks beyond gas turbine blade aerodynamics.

Keywords: gas turbine blades, engineering design optimization, serverless containers, cloud computing, surrogate models, machine learning, MLOps

Citation: Zhemelev G.A., Drobintsev P.D. A software system for surrogate-based prototyping of gas turbine blades using serverless containers in the cloud. Computing, Telecommunications and Control, 2025, Vol. 18, No. 2, Pp. 120–136. DOI: 10.18721/JCSTCS.18210

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.18210
УДК 004.896

ПРОГРАММНАЯ СИСТЕМА ДЛЯ ПРОТОТИПИРОВАНИЯ ЛОПАТОК ГАЗОВЫХ ТУРБИН С ИСПОЛЬЗОВАНИЕМ СУРРОГАТНЫХ МОДЕЛЕЙ И БЕССЕРВЕРНЫХ КОНТЕЙНЕРОВ В ОБЛАКЕ

Г.А. Жемелев ✉, *П.Д. Дробинцев*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ wws.dev@gmail.com

Аннотация. Оптимизация конструкции лопаток газовых турбин – это сложная мультидисциплинарная задача, требующая ресурсоемких физических расчетов. Для их выполнения применяют множество инженерных программных пакетов, часто вместе с суррогатными моделями машинного обучения в целях быстрого прототипирования. Однако на данный момент эффективная интеграция широкого спектра программного обеспечения (ПО) в цикле оптимизации остается актуальной проблемой ввиду трудоемкости установки и настройки компонентов, привязки к конкретным поставщикам ПО, а также недостатка масштабируемости и автоматизации вычислительных конвейеров. В данной статье предлагается оригинальная архитектура системы, основанная на использовании облачных сервисов и контейнеризованных компонентов с целью построения гибких вычислительных конвейеров для инженерной оптимизации. Предлагаемое решение включает в себя применение бессерверных вычислений на контейнерах и асинхронный обмен сообщениями, что вместе с использованием типовых облачных ресурсов позволяет обеспечить масштабируемость, переносимость и устойчивость системы. Кроме того, в ней применяется подход MLOps для эффективной организации жизненного цикла суррогатных моделей, что повышает качество и повторяемость результатов машинного обучения. Предложенное решение превосходит существующие благодаря простоте интеграции разнообразного ПО в цикле оптимизации и простоте в установке для пользователей, а также минимизирует зависимость от конкретных поставщиков за счет использования только открытого и свободно распространяемого ПО и стандартных облачных ресурсов. Проведенные эксперименты по оптимизации аэродинамики лопаток газовых турбин позволили убедиться в масштабируемости и отказоустойчивости системы, а также в ее применимости для быстрого прототипирования с использованием суррогатных моделей. Более того, гибкость разработанной системы и расширяемость ее архитектуры открывают возможности по применению предложенного решения и в других задачах инженерной оптимизации, не ограничиваясь проектированием лопаток газовых турбин.

Ключевые слова: лопатки газовых турбин, инженерное проектирование и оптимизация, бессерверные контейнеры, облачные вычисления, суррогатные модели, машинное обучение, MLOps

Для цитирования: Zhemelev G.A., Drobintsev P.D. A software system for surrogate-based prototyping of gas turbine blades using serverless containers in the cloud // Computing, Telecommunications and Control. 2025. Т. 18, № 2. С. 120–136. DOI: 10.18721/JCSTCS.18210

Introduction

Blades are key components of a gas turbine: their shape heavily affects the efficiency of extracting useful work from the gas flow and ultimately the performance of the entire energy generation unit. Finding the optimal shape of blades for each turbine stage is a time-consuming procedure that requires complex and resource-intensive calculations that model the physical processes in and around the blades.

The multitude of disciplines involved and the need to optimize hundreds of parameters that define the geometry and other properties of each blade – all add up to the labor and time required to bring new turbine models to market. Therefore, the industry is looking for ways to speed up computations, in particular, by using machine learning (ML) to construct surrogate models [1], as well as to increase the degree of automation of the entire blade prototyping process by integrating a variety of software into the continuous optimization cycle. This, in turn, leads to the need to organize the interaction of computer systems that participate in the design optimization of gas turbine blades. Fig. 1 shows a simplified workflow of MultiDisciplinary Optimization (MDO) of a gas turbine blade. Arrows indicate data flows and "P" blocks stand for optional postprocessing routines for each stage.

We can divide the depicted systems into two groups: inside and outside the optimization loop. Steps inside (on a gray background in Fig. 1) form a pipeline, which may include parallel steps. Surrogate models usually replace the pipeline by approximating functions that map design variables into objectives and constraints.

Surrogate models, also known as meta-models or meta-functions, are mathematical models that approximate the behavior of a complex system or function in order to speed up computation or simplify analysis by replacing the original model in relevant problems [2, 3]. Typically, the construction of surrogate models is based on experimental data and is implemented using ML techniques[1] [2–4]. Computational Fluid Dynamics (CFD), Conjugate Heat Transfer (CHT) and Finite Element Analysis (FEA) are perfect examples of disciplines involving long and costly computations, for which surrogate modeling can be used in the process of optimizing blade shapes in terms of their aerodynamics [5]. In that case surrogate models usually take blade design variables as input [6, 7], but it is not mandatory, and 3D blades shape (e.g., represented as a mesh) may be passed directly to a surrogate model, if its architecture is tailored to such inputs, as in [8]. It is up to the optimizer to decide on when to use a surrogate model and when to run the whole pipeline.

The first step of the pipeline usually involves calling the API of a Computer-Aided Design (CAD) system or some other shape generation software (e.g., a generative model as described in [9]) to produce a 3D shape of the blade out of the passed design variables values. Then, follows the meshing step required for further physics calculations like CFD, CHT and/or FEA. Every step may have some post-processing routine to adjust, validate and/or extract results relevant to a specific task.

In practice every step in the design optimization cycle is done using specialized software. This leads to the fact that all steps may have different operating system (OS) requirements, dependencies for third-party libraries and execution environments needed to run the software. The multitude of various technologies involved poses a certain challenge to achieve seamless integration of the steps and effective interaction of the systems inside and outside of the optimization cycle. Also, it is often desirable to avoid vendor lock-in and support proper deployment of the resulting system so that engineers can easily utilize powerful cloud-compute environments to solve optimization tasks submitted from regular PCs or laptops.

The above-mentioned characteristics are often missing in solutions described in literature. For example, a surrogate-based integrated framework by A. Benaouali and S. Kachel [10] is based on SIEMENS NX, ICEM CFD and ANSYS FLUENT software and tailored to corresponding data formats. That makes it tightly coupled to the vendors of those systems and hard to setup, as any potential user would have to install all the required software on their PC together with GRIP and Tcl/Tk tools that are used in the framework for steps integration. In the updated solution [11], the authors enriched the framework's possibilities by supporting multidisciplinary optimization, but that resulted in an even wider set of software to install in order to use that framework, so that all the scripts that do the integration and automation work can run (e.g., a user would need to install MATLAB just to make fluid structure interpolation).

[1] What is Surrogate Model. Available: https://www.deepchecks.com/glossary/surrogate-model (Accessed 18.12.2024)

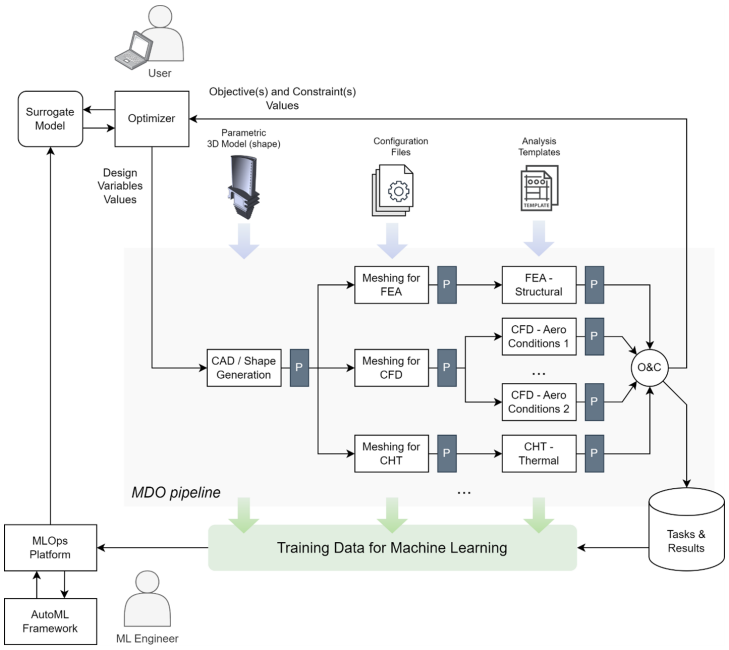

Fig. 1. Simplified workflow of MDO of a gas turbine blade

Another existing solution for surrogate-based design optimization, DADOS [12], is cloud-based, which makes it much easier to use, but on the other hand, it is not an integrated solution: DADOS automates creation and use of surrogate models but does not provide a full optimization cycle with CFD and/or other physics computations. It is expected from a user to upload the computation results to the web-interface of DADOS via Excel files. This fact significantly limits the applicability of the overall solution, because it involves manual steps, and then, surrogate models cannot be run in parallel or within the optimization cycle that involves resource-intensive tasks like CFD.

Even industrial-grade solutions, like HEEDS[2], while having a lot of support for integration with other products, still imply they all are installed on a user's PC, which makes it challenging to run a complex optimization pipeline, as it requires a user, who is typically a mechanical engineer, not an IT specialist, to setup a lot of software and custom scripts with all their dependencies that may have conflicts during installation, may be incompatible with the user's OS etc. This makes creating an integrated solution a challenging task, and while some researchers [5] have successfully built surrogate-based optimization pipelines using HEEDS and their own ML-models, that software is neither portable nor flexible enough for usage in scenarios and environments other than those described in the original paper [5], and does not support cloud deployments. A way to solve these problems is presented in this paper.

Another important part of the surrogate-based design optimization, that is missing in the known solutions, is proper lifecycle management for custom ML models, designed by an ML engineer and/or some AutoML framework. It includes a model registry and datasets storage that support versioning, linked to experiments results and history, as well as serving the learnt models in containers and/or on "as a service" basis. Lack of these characteristics forms technical debt for ML-based systems that lead to big maintenance costs [13]. In response to this problem, the MLOps paradigm has recently emerged. As stated in [14], incorporating MLOps practices is essential for bringing any ML-based solution to a production-grade quality level.

Last but not least, to avoid vendor lock-in and modern challenges of using foreign commercial software in Russia, it is important to make use of open-source and free technologies and software as much as possible, when designing new software solutions, as well as domestic providers for cloud services.

[2] Simcenter HEEDS. Available: https://plm.sw.siemens.com/en-US/simcenter/integration-solutions/heeds (Accessed 25.12.2024)

Methods

Containerization and Clouds

One of the key aspects of the suggested solution is extensive use of containers. A container is a unit of software that packages application code together with all required dependencies (libraries, execution environment etc.) and is managed by a container engine[3]. That engine, like a hypervisor for virtual machines (VMs), isolates containerized applications from the host OS, enabling the portability to run them across various infrastructures that support containers, including clouds and regular PCs[4]. The key difference between a container and a VM is that the former does not start a complete OS on top of the host system, but shares a kernel with it while keeping isolation in the user space[5] [15]. In addition, containers can be restricted in resources allocated to them, which is mainly achieved by using the cgroup mechanism[6]. The lightweight nature of containers, i.e., smaller infrastructural footprint and faster start-up times (compared to VMs) significantly contributes to cost reduction, especially when using cloud resources[7], and improves developer experience and productivity[8] [16]. Containers are widely used in ML and AI companies[9], as well as in various scientific applications, from software engineering research [17–19] to bioinformatics, where containers have been successfully applied to build infrastructure-agnostic human genome sequencing pipelines [20].

In recent years, a new cloud technology has emerged, known as serverless containers[10]. These can be considered as a more powerful kind of cloud lambda functions (that are usually considered, when referring to serverless computing), where a cloud provider accepts a complete Docker image to run instead of just source code to be executed in a selected environment. Still, no server provisioning is required from the user, and all the machine resources needed to run containerized applications are allocated on demand by the cloud service provider on a pay-as-you-go pricing model[11]. In Google's report on the state of DevOps[12], it is highlighted that cloud computing improves productivity of engineers in IT companies. This is valid for serverless containers as well, because these are as easy-to-use as serverless cloud functions and, at the same time, provide much more flexibility as regular containers[13].

In the context of this paper, the most important benefits of containers are portability, environment isolation, simplified deployment and suitability for serverless cloud operation. These characteristics are crucial to build a system that relies on a wide variety of software to run in organized and scalable pipelines launched from a laptop by users, who are engineers, but not IT experts. The portability of

[3] Understanding containers. Available: https://www.redhat.com/en/topics/containers (Accessed 25.12.2024); Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024); Chto takoye konteynerizatsiya: Oblachnaya terminologiya [What is containerization: Cloud terminology]. Available: https://yandex.cloud/ru/docs/glossary/containerization (Accessed 24.12.2024)

[4] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024); Chto takoye konteynerizatsiya: Oblachnaya terminologiya [What is containerization: Cloud terminology]. Available: https://yandex.cloud/ru/docs/glossary/containerization (Accessed 24.12.2024)

[5] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024)

[6] Menage P., Lameter C., Jackson P. Control Groups. Available: https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html (Accessed 25.12.2024)

[7] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024); Chto takoye konteynerizatsiya: Oblachnaya terminologiya [What is containerization: Cloud terminology]. Available: https://yandex.cloud/ru/docs/glossary/containerization (Accessed 24.12.2024); The Total Economic Impact™ of Docker Business. Available: https://www.docker.com/resources/tei-of-docker-business-a-conversation-with-our-cro-webinar/ (Accessed 16.05.2025)

[8] The Total Economic Impact™ of Docker Business. Available: https://www.docker.com/resources/tei-of-docker-business-a-conversation-with-our-cro-webinar/ (Accessed 16.05.2025); Hayzen A. How containers improve the way we develop software. Available: https://www.embedded.com/how-containers-improve-the-way-we-develop-software (Accessed 25.12.2024); Choroomi A. How Kinsta Improved the End-to-End Development Experience by Dockerizing Every Step of the Production Cycle | Docker. Available: https://www.docker.com/blog/how-kinsta-improved-the-end-to-end-development-experience-by-dockerizing-every-step-of-the-production-cycle (Accessed 25.12.2024)

[9] Hype Cycle for Container Technology, 2024. Available: https://www.gartner.com/en/documents/5521795 (Accessed 16.05.2025)

[10] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024); Hype Cycle for Container Technology, 2024. Available: https://www.gartner.com/en/documents/5521795 (Accessed 16.05.2025); Yandex Serverless Containers. Available: https://yandex.cloud/ru/docs/serverless-containers (Accessed 25.12.2024)

[11] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024)

[12] Accelerate State of DevOps 2023. Available: https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf (Accessed 16.05.2025)

[13] Susnjara S., Smalley I. What Is Containerization? Available: https://www.ibm.com/think/topics/containerization (Accessed 25.12.2024); Yandex Serverless Containers. Available: https://yandex.cloud/ru/docs/serverless-containers (Accessed 25.12.2024)

containers allows running the same Docker image in the cloud and locally, thus enabling a quick feedback loop, when developing the image (before releasing it) and flexibility to run lightweight pipelines on a regular PC (while setting up the optimization task and physics calculations templates), before rolling out a full-fledged workload to the cloud.

The Proposed Architecture

Key aspects of the proposed architecture include:

- containerization of software components,
- integration of those in pipelines in the optimization loop,
- storage for all artifacts and settings of pipelines' stages,
- software developed to enable system operation in the cloud,
- utilization of serverless containers in the cloud,
- using Docker Compose when running locally or on VMs,
- AutoMLOps for maintaining surrogate models' lifecycle.

The proposed system's serverless operation mode requires the cloud environment to support the well-known APIs for Simple Storage Service (S3), Simple Queue Service (SQS) and DynamoDB in document mode, as well as the ability to store Docker images and start containers by triggers from SQS. This set was originally introduced by Amazon Web Services (AWS), but now it is available both in foreign cloud providers (Amazon Web Services, Google Cloud Compute, Microsoft Azure) and in domestic solutions, such as Yandex Cloud[14], as it is a de facto standard in this industry. The serverless mode of operation helps to achieve high scalability of the system, provides separation of computing resources and data storage, and gives the opportunity to use built-in monitoring and information security tools provided by the cloud offering.

Fig. 2 shows the general scheme of the proposed system architecture, using the Amazon Web Services notation.

A more detailed structural scheme of the pipeline with indication of used technologies is shown in Fig. 3. In this case, the components are mapped to Yandex Cloud services, since the system was implemented using the resources of this cloud provider.

The system uses Dakota[15] as an optimizer, an open-source solution that includes many optimization algorithms, including those that support global optimization using surrogate models. In addition to optimization, Dakota has a design of experiments (DoE) functionality, which allows sampling of blades parameters to achieve uniform coverage of the search space (e.g., using the Latin hypercube algorithm), which is useful for training surrogate ML-models.

At each iteration in the optimization process, Dakota invokes the Cloud Task Runner program, which is written by the authors in the Go language. This program is responsible for generating a task and starting one run of the pipeline. It writes to the database the information necessary to launch the pipeline, including the values of variables describing the blade and configuration files for the components, as well as the sequence of their execution. Each component is responsible for its own stage. The hierarchy "task – run – stage" is reflected in the structure of S3 storage (Fig. 3), which is used to store all artifacts produced by the components of the pipeline, and for data exchange between them (together with SQS messaging). The artifacts are divided into four groups: configuration files, input files, output files and additional files. The need for particular files is determined by each specific component. An example of additional files is an archive with complete CFD calculation results, which often has a size of hundreds of megabytes or more depending on the mesh detail and the number of steps in the calculation. A cheaper class of S3 storage (standard infrequent access) is used for additional files.

Each component is deployed as a Docker container, which includes the component's software and a program developed by the authors called Cloud Connector. This program is written in the Go language

[14] Yandex Serverless Containers. Available: https://yandex.cloud/ru/docs/serverless-containers (Accessed 25.12.2024)
[15] Dakota. Available: https://dakota.sandia.gov (Accessed 11.01.2025)

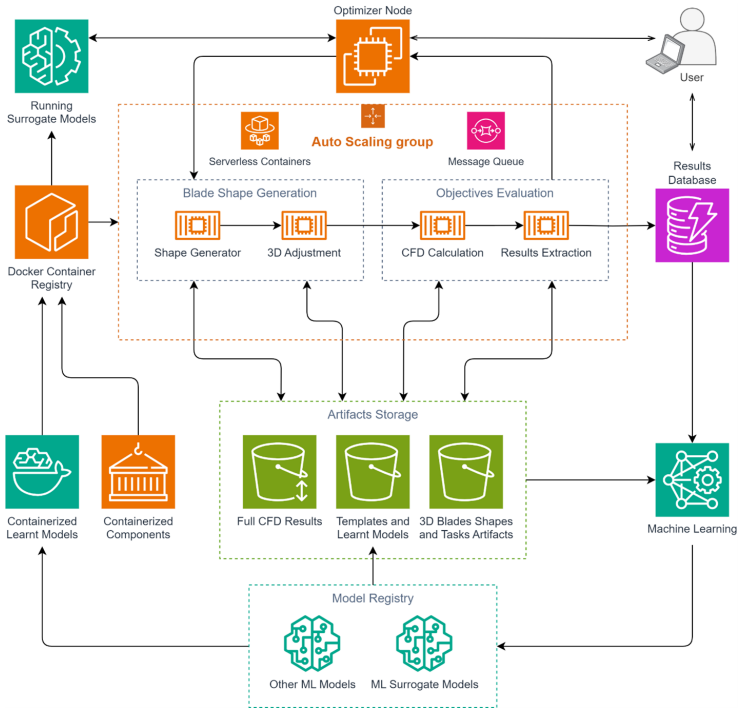

Fig. 2. The proposed system's architecture for cloud deployment

and is responsible for universal interaction of components with cloud services: SQS, S3, DynamoDB – through the authors' library Cloud Task Registry, that provides an application programming interface Task Registry API and is used both in the Cloud Connector and Cloud Task Runner programs. Together these programs and the library make up the software complex named Cloud Optimization Suite.

After solving an optimization problem formulated by the user in the Dakota interface, all configuration, input and output files and other artifacts for each of the stages of each run remain recorded in the S3 storage. This includes, among other things, 3D shapes of the generated blades and complete CFD calculation results. This makes each step fully reproducible: a user can download the relevant files, run the component responsible for the stage on a local machine and study its operation in detail for specific parameters or take the files of interest for further work with them. Similar functionality of data storage is implemented in the local operation mode using Docker Compose, but in this case, data is stored on disk in the directory specified by the user, and the cloud functionality is not used, which allows working with the proposed system locally and fully independent of cloud providers, or on VMs.

Training of Surrogate Models

Parameters of generated blades' shapes along with CFD results and other artifacts essentially form datasets for surrogate models training. This feature of the proposed architecture enables straightforward integration with AutoML frameworks. These allow searching for best regression models automatically. Alternatively, ML-engineers may design more sophisticated models within the MLOps framework, as was described earlier. In combination, this leads to an AutoMLOps solution. In the proposed system, ClearML open-source AI platform[16] is used for that purpose together with Auto-Sklearn 2.0 framework [21] for AutoML support. ClearML has a free version (Apache-2.0 license) and uses exclusively open-source components, allowing integration with a lot of other frameworks and services. Since ClearML supports integration with AWS S3, the training data can be used directly from the artifacts storage (Fig. 2).

Using the MLOps platform, the following ML pipeline is designed:

1. Load new training data from the S3 storage.

[16] ClearML Open Source Platform. Available: https://clear.ml/ai-platform (Accessed 11.01.2025)

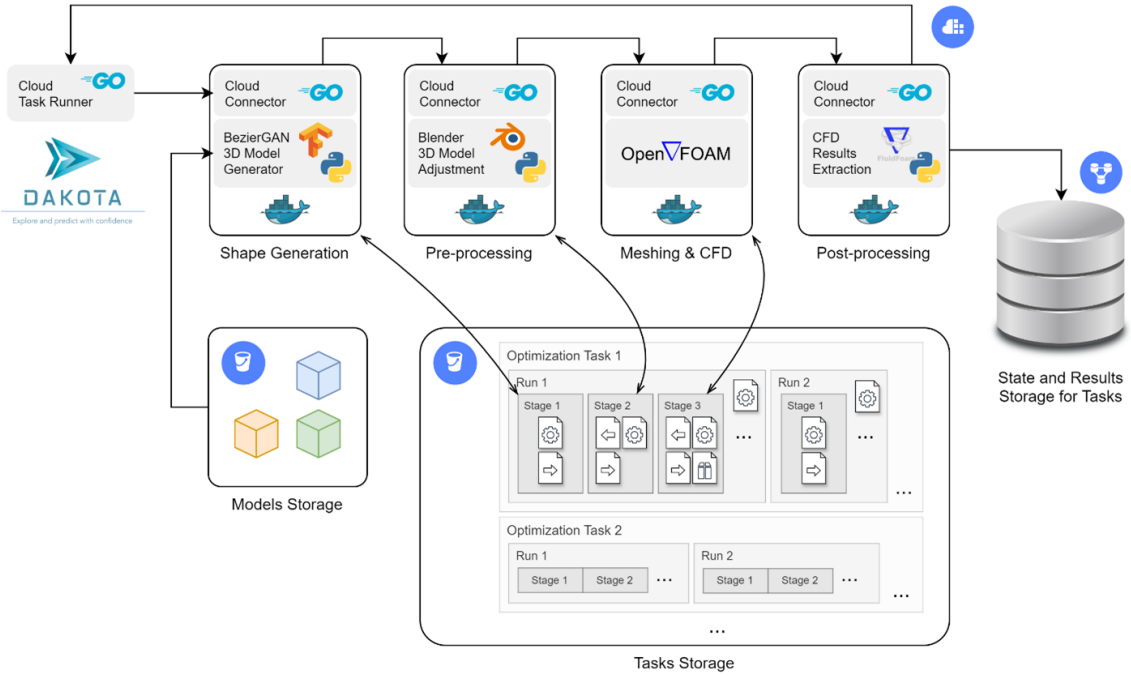

Fig. 3. Structure of the pipeline and technologies used

2. Preprocess the data if needed (e.g., to extract some extra features).
3. Call the AutoML framework for training of a surrogate model.
4. Test the model performance.
5. Optionally, perform hyperparameters tuning.
6. Store the model and assign a version to it.

Pipeline orchestration is performed by the ClearML server (Fig. 4), and the training itself is done by ClearML agents, which are deployed on machines that have all required hardware resources (CPUs and/or GPUs).

Communication with the agents is done via tasks queues. The trained models are stored in the model storage (e.g., in an S3 bucket). In order to perform inference using saved models, ClearML provides a serving solution. It allows to expose HTTP endpoints to accept inference requests and returns corresponding responses after delegating the processing to a configured serving engine (ClearML offers its own CPU-based engine and also supports Triton[17] by Nvidia for GPU-enabled hardware). In the proposed system, the surrogate model interface of Dakota is used to make such inference calls, passing blades geometric parameters that are subject to optimization.

By the point, when surrogate models training is performed, a large number of expensive calculations is typically done, and CFD results are obtained for a multitude of blade shapes that may be confidential as a company asset, as well as surrogate models themselves. Thus, we obtain a large amount of sensitive information in a single step, possibly even on a single VM, which necessitates additional protection. This topic was covered in [22–24] where the Trusted AutoML task was formalized and addressed from performance and cybersecurity perspectives, including but not limited to the application for training surrogate models on gas turbines' data.

Implementation

The Cloud Optimization Suite software is implemented using the Go programming language (Golang). The choice of the language was made to fulfill the need to minimize the footprint of the Cloud

[17] Triton Inference Server. Available: https://developer.nvidia.com/triton-inference-server (Accessed 18.01.2025)

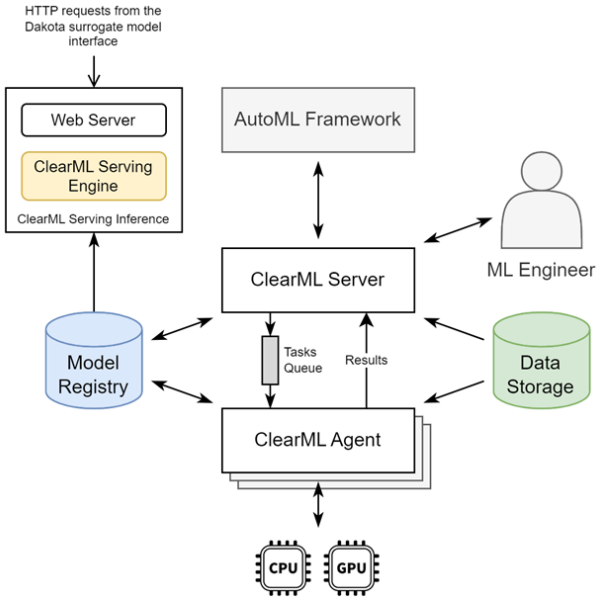

Fig. 4. Detailed view on the MLOps platform used in the system

Connector on container images of the pipeline stages components' software and simplify integration. Programs in Go are compiled into natively executable binaries and can be statically linked with all used libraries, thus making the program independent from other system libraries. As a result, installation of the Cloud Connector is just copying one executable file into a Docker container. Golang provides a garbage collector, which simplifies memory management and a set of useful concurrent programming mechanisms (goroutines, channels etc.), which are particularly helpful when dealing with asynchronous calls.

One of Golang's distinctive features is absence of an exception facility in the language (i.e., there is no control structure associated with error handling), and errors are handled in the same way as regular variables values. The language authors stand for that decision, claiming that "explicit error checking forces the programmer to think about errors – and deal with them – when they arise"[18]. Although at the moment there is no scientific evidence that software developed using Go has better quality than that written in other languages [25], explicit error handling has proven to be useful in the process of developing the Cloud Optimization Suite and improved its robustness by enforcing consideration of many exceptional scenarios beforehand.

The Cloud Optimization Suite source code is available at GitHub[19] under the Apache-2.0 permissive license and consists of three Go modules: Cloud Task Registry, Cloud Connector and Cloud Task Runner. The first one is a library for communication with cloud resources and two others are applications: Cloud Task Runner is placed at the machine, where the Dakota optimizer is installed, and Cloud Connector is appended to each components' Docker image and used as an entry point (with additional arguments, like a stage name, the main component executable path, DynamoDB document API URL etc., provided either at Docker build time or via environment variables).

Cloud Task Runner populates the database tables with a new record for a task run and corresponding records for its stages according to a configuration file and then submits the task for processing. It also copies configuration files for the stages' components to an S3 bucket according to the structure shown in Fig. 3. After submitting the task for execution, the Cloud Task Runner program waits for the task run completion via long polling of the final SQS queue, while the task run's Universally Unique Identifier

[18] Pike R. Go at Google: Language Design in the Service of Software Engineering. Available: https://go.dev/talks/2012/splash.article (Accessed 09.02.2025)
[19] Cloud Optimization Suite. Available: https://github.com/wndrws/cloud-optimization-suite (Accessed 08.02.2025)

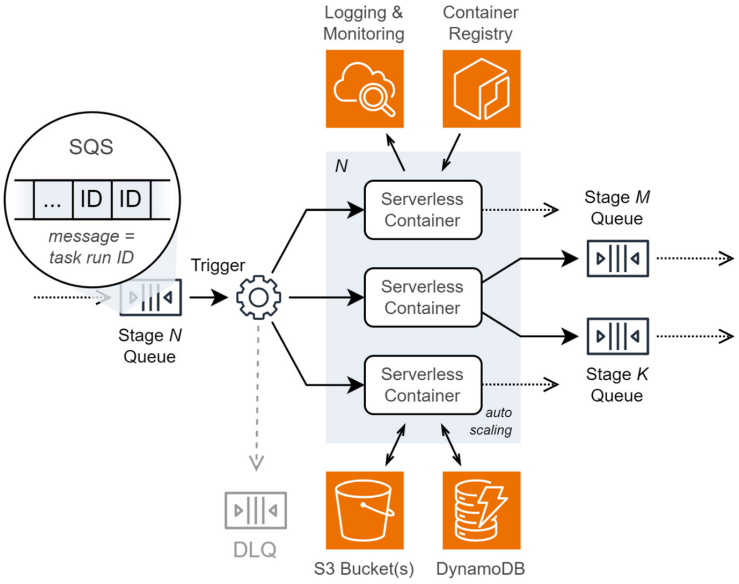

Fig. 5. Functional diagram of a stage within the pipeline

(UUID) makes its way through the pipeline. The generic functional diagram of a pipeline stage is given in Fig. 5. Arrows with big heads represent message flow, and arrows with smaller heads stand for data flow (Docker images, files, database communication, logs etc.).

Each pipeline stage is associated with one software component that implements the stage's functionality and deployed as a set of serverless container instances. There may be none, one or many container instances at each stage in any given moment in time depending on the number of messages in processing and scaling configuration in the cloud console. Yandex Cloud uses triggers to monitor SQS queues, and when a new message arrives, it creates a new container instance and passes the message to it; then, it shuts down the instance if all messages have been processed and the queue is empty. When a serverless container instance finishes a task run processing, it uploads output files to S3 (using the Cloud Connector) and sends the run's UUID to the SQS queues of the next adjacent stage(s). All logs written by the component are collected by the cloud provider and managed using the Cloud Logging service. Thus, the system administrator can see logs from all containers in one place in the cloud console together with monitoring information, like CPU load and RAM consumption of the software in serverless containers.

When a message from an SQS queue is passed to a serverless container, it goes to a Cloud Connector's handler. The algorithm of the Cloud Connector program can be briefly formulated as follows (for each stage):

1. Cloud Connector waits for messages from the queue in SQS. The incoming message contains a run UUID and initiates the start of the stage.

2. After requesting information about the task and the stage from DynamoDB, Cloud Connector downloads the configuration and input files of the stage from S3.

3. Using the configured component start command, Cloud Connector creates the corresponding subprocess and waits for it to complete. In parallel, Cloud Connector makes periodic queries to DynamoDB, checking the status of the task to see if it has been canceled. If the task has been canceled, a termination signal (SIGTERM) is sent to the subprocess.

4. Based on the value of the subprocess return code, Cloud Connector determines the fact of successful or unsuccessful execution of the pipeline stage: a non-zero return code means that an error occurred.

5. If the component startup was successful, Cloud Connector reads the output file (if the path was specified) and loads it into S3. A message with the run ID is then sent to the SQS queue(s) defined for the stage(s) immediately following the current stage.

6. If this stage is the last stage in the run (i.e., has no next stages in the task pipeline configuration), the output file is read as an associative array of objectives (and/or constraints) and their values and written as the task run results to DynamoDB.

7. If paths to additional artifacts are configured for the step, these folders and/or files are compressed into an archive and uploaded to S3.

If an error occurs during a Cloud Connector operation, the corresponding status is set for this stage of the task, the Cloud Connector terminates, and then the container is restarted by means of the cloud provider. If the limit of restarts is exhausted, the task run ID is sent to a special Dead Letter Queue, which is not related to the pipeline stages, but can be used for debugging.

After the last step of the run is completed, its identifier is sent to a special SQS queue, from which it is read by the Cloud Task Runner program. Then, it outputs a report listing all the attributes, status and execution time of each stage and the run as a whole. The cycle repeats until a convergence condition (configured in the optimizer) is reached or the task is cancelled.

The data model of DynamoDB tables is shown in Tables 1 and 2: one table is used to store information about task runs, and the other one is for their stages at each run.

Table 1

DynamoDB data model – the task runs table

Keys	Name	Type	Explanation
PK	task_id	string	Identifier of the task
SK, GSI PK	run_uuid	string	Identifier of the task run
	parameters	map<string, number>	Design variables values for the run
	results	map<string, number>	Objectives and constraints values
	task_definition	string	The task configuration from Dakota
	creation_time	number	Time when the task was created
	status	string	Status of the task run

There is a one-to-many relationship between the stored entities: each task run record is associated with many task stage records. Saving timestamps of start and completion of each stage for every task run is useful for monitoring and collecting statistics of the system functioning. The tables are created automatically, when Cloud Task Registry is used with an empty DynamoDB (by the means of migration code implemented in Go).

The DynamoDB keys abbreviations are as follows: PK – partition key, SK – sort key, GSI – global secondary index. Possible statuses in the task runs table are Pending, InProgress, Success, Error, Cancelled, – and possible statuses in the task stages table are Submitted, Finished, Failed and Cancelled.

It is worth noting that the system provides users with an ability to cancel tasks execution, forcing the running containers to interrupt component's software, do the required cleanup and state management and terminate.

Results and Discussion

After implementation, the system was tested in a series of experiments, using Yandex Cloud resources, as well as local Docker Compose deployments. The experiments performed can be grouped as follows:

1. Testing the system operation in the DoE mode with different numbers of parallel running pipelines to assess the system's scalability and correctness of its functioning in both variants of deployment.

2. Checking the system operation in optimization mode to find the gas turbine blade shape that maximizes the aerodynamic efficiency coefficient with and without the use of surrogate models, comparison of the found optima and the time spent.

3. Validation of correctness of CFD results produced using the system from the domain perspective.

Table 2

DynamoDB data model – the task stages table

Keys	Name	Type	Explanation
PK, GSI PK	run_uuid	string	Identifier of the task run
SK	n_ord	number	Ordinal number of the stage
GSI SK	name	string	Name of the stage
	status	string	Status of the task run at this stage
	config	string	Paths to configuration, input and output files of the component at this stage in the S3 bucket (see below)
	input	string	
	output	string	
	t_start_utc	number	Times when the task run processing started and finished at this stage
	t_finish_utc	number	
	executor	string	Execution environment information (e.g., revision of the Docker image used for the run)
	s3_bucket	string	Name of the S3 bucket used
	comment	string	Any additional information from the component
	next	list<string>	List of the adjacent next stages names

The obtained results have shown that the developed system functions correctly both in a single-threaded environment and when running pipelines in parallel (in both deployment modes: serverless and on a single machine via Docker Compose); the integrity of data accessed concurrently is not violated; the system scales vertically according to the computational resources of the processor and horizontally according to the number of available physical cores (Table 3): Hyper-Threading technology does not give advantages, which was revealed during experiments on the Intel Core i7-7700K CPU. When used in serverless mode, horizontal scaling is done by utilizing more instances of serverless containers in parallel within quotas set by the cloud service provider.

Table 3

Time spent on DoE using the suggested system on a PC with 4 physical and 8 logical CPU cores

Concurrent pipelines count	1	2	3	4	5	6	7	8
Total time, s	15970	8655	6617	5830	5840	5713	5894	5730

The use of surrogate models in the Efficient Global Optimization mode led to at least 2.5 times reduction in time for the optimal blade shape search compared to optimization without surrogate models, given the same number of the pipeline runs. The achieved optimal value of the aerodynamic efficiency coefficient was different only by 2.46% in favor of the optimization without surrogate models. In serverless mode the time spent on optimization (in fixed conditions) was 18–36% less compared to running on a VM with the same number of processor cores of the same architecture (Intel Haswell) and the same

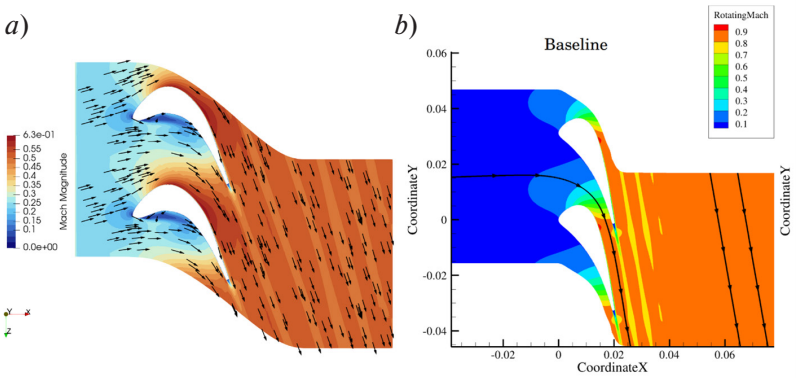

Fig. 6. The velocity field obtained by the authors (*a*) and a reference (*b*) from [26]

clock frequency of 2.1 GHz (2 GB RAM per core allocated by the cloud provider was more than enough for all containerized components).

Validation of CFD results produced using the system was successful according to the set of quantitative checks and qualitative analysis performed by the authors. For example, Fig. 6 shows a comparison of the velocity field (in Mach numbers) obtained by the authors and a similar reference field from [26].

A separate discussion should be given to the architectural advantages of the system. As its components communicate asynchronously via message queues (SQS), which transport only task runs' identifiers, loose coupling is achieved, providing the following benefits:

Fault-tolerance and resilience. If an error occurs during functioning of a pipeline component, the task run processing request returns to the queue and becomes visible for other containers at the stage[20] that can handle the processing request, while the failed serverless container is shut down by the platform and replaced by a new instance if needed for serving the following requests. The recover process does not impact the functioning of other system parts, e.g., parallel pipeline runs continue unaffected, though the erroneous task status change is reflected in the DynamoDB and is visible via serverless containers monitoring. Status tracking of individual task runs also protects the system from using any partly written data that may be left by failed serverless containers. In case a container becomes unresponsive, it also will be terminated by the cloud platform according to the configured timeout (separately for each pipeline stage).

Hot swap of Docker images. When the root-cause of an error is identified in a containerized component and fixed, it is possible to upload the new Docker image and replace the one currently utilized by the serverless container without stopping the pipeline operation (Yandex Cloud gracefully stops the running container instance and starts a new one instead). The same task runs, that were not passing previously, will then be automatically retried using the latest component version[21]. This hot swap support is highly beneficial for lengthy and costly optimization tasks, when restarting from the beginning in case of any error is not affordable.

Flexibility in data exchange formats and software selection. Since in the proposed architecture there are no requirements for data formats used by pipelines components, they can be chosen freely by the implementations and must be agreed only between the neighboring stages (as one stage's output files usually serve as the next stage's inputs). This, together with containerization, enables flexibility in selecting the formats and software used at each pipeline stage, without the need to make any adjustments to the system to maintain compatibility. By abstaining high-level code (i.e., the Cloud Optimization Suite) and the system operation principles from low-level details, the architecture follows the dependency inversion principle [27] and the open-closed principle [28] widely adopted in software engineering [29].

[20] Trigger for Message Queue that sends messages to the Serverless Containers container. Available: https://yandex.cloud/en-ru/docs/serverless-containers/concepts/trigger/ymq-trigger (Accessed 04.02.2025)

[21] Yandex Serverless Containers. Available: https://yandex.cloud/ru/docs/serverless-containers (Accessed 25.12.2024)

Table 4

Comparison of the suggested system and existing solutions

Criteria	Ours	Song et. al., 2023 (DADOS) [12]	Benaouali and Kachel, 2017 [10]	Benaouali and Kachel, 2019 [11]
Integrated solution	Yes	No	Yes	Yes
Data file formats	Arbitrary	Excel	Parasolid, FluentMesh	Parasolid, IGES, FluentMesh, PATRAN, NASTRAN
Software components in the pipeline	Any containerizable (using Docker)	Any (at the user's side)	Siemens NX, ICEM CFD, FLUENT	Siemens NX, ICEM CFD, FLUENT, MSC.PATRAN, MSC.NASTRAN, MATLAB
Environment	Any cloud providing S3, DynamoDB and SQS APIs or Docker Compose	Cloud (China)	Local	Local
Supported surrogate models	Any models provided by Dakota or own ML-models via ClearML MLOps	PRS, RBF, KRG, MLS, SVR, and ANN (feed-forward)	Gaussian RBF	RBF
Optimization algorithms	DIRECT, EGO and others supported by Dakota[22]	In-house + SA, swarm, genetic and other algorithms	Sequential Quadratic Programming	Genetic Algorithm
Supported DoE sampling	LHS, Monte-Carlo, Rank-1 Lattice, Digital Nets (Sobol)	OA, CCD, LHS, OLHS	LHS	Improved LHS
Parallel steps in pipelines	Yes	No	No	Yes
Multi-disciplinary	Yes	No	No	Yes
Publicly available	Yes	Yes (after registration)	No	No
Open source	Yes	No	No	No

Scalability. Using asynchronous communication and cloud services, plus separation of storage and compute makes the system highly scalable. New instances of serverless containers are created automatically by the cloud platform in response to new messages in task queues. As soon as some task runs are finished, unnecessary containers are shut down and don't incur any more costs in accordance with the pay-as-you-go model. More parallel pipelines or parallel branches within one pipeline do not induce resource contention, since the DynamoDB database, SQS queues and S3 buckets (that are used for data exchange and storage), are designed for efficient handling of concurrent loads, and apart from them there are no shared resources that containers can contend for. Vertical scaling is also supported, as serverless containers are configurable in terms of CPU cores and RAM allocated by the cloud provider[23]. Finally, as the system compute resources scale down to zero in absence of tasks, there is no risk of wasting resources due to human factor, as it can be with VMs, which are sometimes unintentionally left powered on for a weekend without any workload, producing unwanted spendings.

[22] Optimization Usage Guidelines. Available: https://snl-dakota.github.io/docs/6.21.0/users/usingdakota/studytypes/optimization.html#opt-usage (Accessed 10.02.2025)

[23] Runtime environment. Available: https://yandex.cloud/en-ru/docs/serverless-containers/concepts/runtime (Accessed 04.02.2025)

Apart from the abovementioned advantages, the system administrator can also make use of cloud services that come together with the Serverless Containers service: monitoring (including performance charts and logging), access and secrets management, billing details etc.

To compare the suggested system with the existing solutions discussed previously, a set of criteria was devised considering the most relevant characteristics within the context of this research, and the comparison results are presented in Table 4. Used abbreviations: PRS – polynomial response surface, RBF – radial basis functions, KRG – kriging, MLS – moving least squares, SVR – support vector regression, ANN – artificial neural network; DIRECT – dividing rectangles, EGO – efficient global optimization, SA – simulated annealing; LHS – Latin hypercube sampling, OLHS – optimal LHS, CCD – central composite design, OA – orthogonal arrays.

From that table it becomes even more apparent that the suggested system is more generic and gives more possibilities to build engineering optimization pipelines from various components without the need to install them on every user's PC, plus the support for ML-models that can use full power of modern ML frameworks, AutoML and MLOps for proper models' lifecycle management.

Last but not least, the system does not rely on any closed-source components and avoids cloud vendor lock-in as far as possible by sticking to the most common set of cloud services APIs, namely the S3, SQS and DynamoDB in document mode, which are available in many Russian and foreign cloud providers. All the open-source and free software used have enterprise-friendly licensing (i.e., no copyleft), which is crucial for open-source software clearing purposes and enables commercial use of the system.

Conclusions

In this paper a computer system architecture was proposed together with its implementing software for rapid prototyping of gas turbine blades, which allows prediction of their physical properties by given geometric parameters. This is an integrated cloud-based solution, though flexible enough to be run locally on a regular PC, without the need for complex environment setup for users (mechanical engineers). Despite the fact that the system was described in application to gas turbine blades aerodynamics optimization, it is not restricted to that domain and can be readily used in design optimization tasks for other kinds of industrial objects and physics disciplines.

The next steps in this research direction are to support the hybrid mode of operation, in which the system will utilize serverless containers simultaneously with dedicated VMs for long-running evenly distributed workloads, and to support non-parametric surrogate models that take 3D blade geometry as input [8, 30] to investigate their performance and generalization potential.

REFERENCES

1. **Hammond J., Pepper N., Montomoli F., Michelassi V.** Machine learning methods in CFD for turbomachinery: A review. *International Journal of Turbomachinery, Propulsion and Power*, 2022, Vol. 7, No. 2, Art no. 16. DOI: 10.3390/ijtpp7020016

2. **Martins J.R.R.A., Ning A.** *Engineering Design Optimization*. Cambridge, UK: Cambridge University Press, 2022. DOI: 10.1017/9781108980647

3. **Jiang P., Zhou Q., Shao X.** *Surrogate Model-Based Engineering Design and Optimization*. Singapore: Springer, 2020. DOI: 10.1007/978-981-15-0731-1

4. **Xu L., Jin S., Ye W., Li Y., Gao J.** A review of machine learning methods in turbine cooling optimization. *Energies*, 2024, Vol. 17, No. 13, Art. no. 3177. DOI: 10.3390/en17133177

5. **Zhang C., Janeway M.** Optimization of turbine blade aerodynamic designs using CFD and neural network models. *International Journal of Turbomachinery, Propulsion and Power*, 2022, Vol. 7, No. 3, Art. no. 20. DOI: 10.3390/ijtpp7030020

6. **Kulfan B.M.** Universal parametric geometry representation method. *Journal of Aircraft*. 2008, Vol. 45, No. 1, Pp. 142–158. DOI: 10.2514/1.29958

7. **Zhemelev G.** Parameterized 3D representation of gas turbine blades for machine learning applications. *2024 Wave Electronics and its Application in Information and Telecommunication Systems* (*WECONF*), 2024, Pp. 1–6. DOI: 10.1109/WECONF61770.2024.10564621

8. **Mou S., Bu K., Ren S., Liu J., Zhao H., Li Z.** Digital twin modeling for stress prediction of single-crystal turbine blades based on graph convolutional network. *Journal of Manufacturing Processes*, 2024, Vol. 116, Pp. 210–223. DOI: 10.1016/j.jmapro.2024.02.054

9. **Zhemelev G.A.** Automatic synthesis of 3D gas turbine blades shapes using machine learning. *Information Security Problems. Computer Systems*, 2024, Vol. 59, No. 2, Pp. 152–168. DOI: 10.48612/jisp/dx8x-2he5-tffd

10. **Benaouali A., Kachel S.** A surrogate-based integrated framework for the aerodynamic design optimization of a subsonic wing planform shape. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 2017, Vol. 232, No. 5, Pp. 872–883. DOI: 10.1177/0954410017699007

11. **Benaouali A., Kachel S.** Multidisciplinary design optimization of aircraft wing using commercial software integration. *Aerospace Science and Technology*, 2019, Vol. 92, Pp. 766–776. DOI: 10.1016/j.ast.2019.06.040

12. **Song X., Wang S., Zhao Y., Liu Y., Li K.** DADOS: A cloud-based data-driven design optimization system. *Chinese Journal of Mechanical Engineering* (*English Edition*), 2023, Vol. 36, Art. no. 34. DOI: 10.1186/s10033-023-00857-x

13. **Sculley D., Holt G., Golovin D., Davydov E., Phillips T. et al.** Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 2015, Vol. 28, pp. 2503–2511.

14. **Kreuzberger D., Kühl N., Hirschl S.** Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 2023, Vol. 11, Pp. 31866–31879. DOI: 10.1109/ACCESS.2023.3262138

15. **Bentaleb O., Belloum A.S.Z., Sebaa A., El-Maouhab A.** Containerization technologies: taxonomies, applications and challenges. *The Journal of Supercomputing*, 2022, Vol. 78, No. 1, Pp. 1144–1181. DOI: 10.1007/s11227-021-03914-1

16. **Koskinen M., Mikkonen T., Abrahamsson P.** Containers in software development: A systematic mapping study. *Product-Focused Software Process Improvement* (*PROFES 2019*), 2019, Vol. 11915, Pp. 176–191. DOI: 10.1007/978-3-030-35333-9_13

17. **Kim B.S., Lee S.H., Lee Y.R., Park Y.H., Jeong J.** Design and Implementation of cloud docker application architecture based on machine learning in container management for smart manufacturing. *Applied Sciences*, 2022, Vol. 12, No. 13, Art. no. 6737. DOI: 10.3390/app12136737

18. **Bobunov A.** Using containerization to simplify and accelerate testing processes in financial organizations. *International Journal of Humanities and Natural Sciences*, 2024, Vol. 95, No. 8–1, Pp. 113–117. DOI: 10.24412/2500-1000-2024-8-1-113-117

19. **Cito J., Ferme V., Gall H.C.** Using Docker containers to improve reproducibility in software and web engineering research. In: *Web Engineering* (eds. A. Bozzon, P. Cudre-Maroux, C. Pautasso), 2016, Vol. 9671, Pp. 609–612. DOI: 10.1007/978-3-319-38791-8_58

20. **Kadri S., Sboner A., Sigaras A., Roy S.** Containers in bioinformatics: Applications, practical considerations, and best practices in molecular pathology. *Journal of Molecular Diagnostics*, 2022, Vol. 24, No. 5, Pp. 442–454. DOI: 10.1016/j.jmoldx.2022.01.006

21. **Feurer M., Eggensperger K., Falkner S., Lindauer M., Hutter F.** Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *arXiv:2007.04074*, 2020. DOI: 10.48550/arXiv.2007.04074

22. **Bezzateev S.V., Fomicheva S.G., Zhemelev G.A.** Trusted automatic machine learning in the operation of digital twins. *T-Comm*. 2024, Vol. 18, No. 7, Pp. 44–55. DOI: 10.36724/2072-8735-2024-18-7-44-55

23. **Bezzateev S.V., Fomicheva S.G., Zhemelev G.A.** Techniques for accelerating algebraic operations in agent-based information security systems. *2023 Wave Electronics and its Application in Information and Telecommunication Systems* (*WECONF*), 2023, Pp. 1–6. DOI: 10.1109/WECONF57201.2023.10147978

24. **Bezzateev S.V., Zhemelev G.A., Fomicheva S.G.** Research on the performance of AutoML platforms under confidential computing. *Information Security Problems. Computer Systems*, 2024, Vol. 61, No. 3, Pp. 109–126. DOI: 10.48612/jisp/abff-du38-v739

25. **Ray B., Posnett D., Devanbu P., Filkov V.** A large-scale study of programming languages and code quality in GitHub. Communications of the ACM, 2017, Vol. 60, No. 10, Pp. 91–100. DOI: 10.1145/3126905

26. **Aissa M.H.** GPU-accelerated CFD Simulations for Turbomachinery Design Optimization. Delft, Netherlands: Delft University of Technology, 2018. DOI:10.4233/UUID:1FCC6AB4-DAF5-416D-819A-2A7B0594C369

27. **Martin R.C.** OO design quality metrics: An analysis of dependencies. Report on Object Analysis and Design, 1995, Vol. 2.

28. **Meyer B.** Object-oriented Software Construction (Prentice-Hall International series in computer science). NJ: Prentice-Hall, 1988.

29. **Martin R.C.** Design principles and design patterns. Object Mentor SOLID Design Papers Series, 2000, Vol. 1, No. 1, Pp. 1–34.

30. **Cao J., Li Q., Xu L., Yang R., Dai Y.** Non-parametric surrogate model method based on machine learning with application on low-pressure steam turbine exhaust system. Journal of the Global Power and Propulsion Society, 2022, Vol. 6, Pp. 165–180. DOI: 10.33737/jgpps/151661

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Zhemelev Georgiy A.
Жемелев Георгий Алексеевич
E-mail: wws.dev@gmail.com
ORCID: https://orcid.org/0000-0001-7126-6787

Drobintsev Pavel D.
Дробинцев Павел Дмитриевич
E-mail: drob@ics2.ecd.spbstu.ru
ORCID: https://orcid.org/0000-0003-1116-7765

Submitted: 19.02.2025; Approved: 17.04.2025; Accepted: 29.04.2025.
Поступила: 19.02.2025; Одобрена: 17.04.2025; Принята: 29.04.2025.