FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Laura Battistella Fiorini

Creation and configuration of hybrid machine learning models for process optimization in the series production of complex optics

Aachen, Germany
2023

Laura Battistella Fiorini

Creation and configuration of hybrid machine learning models for process optimization in the series production of complex optics

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Marcelo Ricardo Stemmer, Dr.
Co-supervisor: Henrik Heymann, M.Sc. M.Eng.

Aachen, Germany
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fiorini, Laura Battistella
Creation and configuration of hybrid machine learning models for process optimization in the series production of complex optics / Laura Battistella Fiorini ; orientador, Marcelo Ricardo Stemmer, coorientador, Henrik Heymann, 2023.
74 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Modelos híbridos. 3. Aprendizado de máquina. 4. Conformação a quente. 5. Modelagem de vidro. I. Stemmer, Marcelo Ricardo. II. Heymann, Henrik. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Laura Battistella Fiorini

Creation and configuration of hybrid machine learning models for process optimization in the series production of complex optics

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, June 30th, 2023.

ACKNOWLEDGEMENTS

First, I would like to express my deepest gratitude to my parents, Daisy Maria Battistella Fiorini and Ademar Sérgio Fiorini, and my close family, for their persistent support and encouragement throughout my academic journey. Their belief in me has been the foundation of my success.

I would like to thank the BRAACHEN program coordination and Fraunhofer Institute for Production Technology IPT for the opportunity of this internship and all the precious knowledge that I have acquired during this year in Germany. Especially to Henrik Heymann, my local supervisor, for his guidance, expertise, and mentorship throughout the evolution of this work.

To professor Marcelo Ricardo Stemmer, my academic advisor, thank you for the valuable insights and feedback provided in different stages of development.

To all the friends I made during this year in Germany, thank you for making this experience even more memorable. Especially to my friends from the 4th floor HiWi room, Fabian, Fernando, Gustavo, Henrique, Luiza, Michel, Pedro, and Victor, I am extremely grateful for all the times we have helped each other and expressed our concerns, and, mostly, for sharing many laughs and happy memories.

Finally, I would like to thank my friends from UFSC: Gustavo, Juliana, Leonardo, Maurici, Maurício, Rhanna, Vinícius, and Yuri. I am deeply grateful for the cheerful moments we have spent together, for the inspiration you provide me, and also for the support we always gave each other in difficult times.

DISCLAIMER

Aachen, June 30th, 2023.

As representative of the Fraunhofer Institute for Production Technology IPT in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (*"Trabalho de Conclusão de Curso"*), in accordance with the *Resolução Normativa n$^\circ$126/2019/CUn*.

Henrik Heymann

Digital unterschrieben von Henrik Heymann
Datum: 2023.07.19 20:31:30 +02'00'

Henrik Heymann, M.Sc. M.Eng.
Fraunhofer Institute for Production Technology IPT

ABSTRACT

In recent years, sophisticated technical optics and thin glasses have seen increased usage in German growth markets such as the automotive, electronics, and tech industries. Due to the complexity of these products, their serial production is susceptible to irregularities; therefore, procedures for quality control need to be employed. There is a special concern for quality control during the hot forming of thin glass, given that it is difficult to measure details about this process using sensors. There are alternatives to predict the outcome using machine learning models; nevertheless, they usually demand trained data scientists and a significant amount of high-quality data, which are challenging requirements for small and medium enterprises to meet. This thesis presents an approach using hybrid models that combine machine learning and physics-based models and have fewer prerequisites regarding historical data and specialized personnel. This solution aims to develop a framework that automatically creates and configures a hybrid model structure to predict the shape deviation of glass during the hot forming process. To meet this objective, initial studies were conducted to explore the state of the art in the applications of hybrid machine learning models and to understand the different layout possibilities. Subsequently, the requisites of the project were further analyzed, and a workflow was created to describe the process of generating hybrid models. Thereafter, a computer program was created using Python to put this logic into practice and its validity was checked by applying it to the use case.

Keywords: Hybrid models. Machine learning. Hot forming. Glass molding.

RESUMO

Nos últimos anos, objetos óticos sofisticados e vidros finos têm sido usados cada vez mais por mercados alemães em crescimento, como os setores automotivo, de eletrônica e tecnologia em geral. Devido à complexidade, a produção em série desses produtos está suscetível a irregularidades e, portanto, faz-se necessário o emprego the procedimentos de controle de qualidade. Existe uma preocupação especial em relação ao processo de conformação a quente de vidros finos, por causa da dificuldade em medir este procedimento utilizando sensores. Existem alternativas para prever o resultado dessa operação utilizando aprendizado de máquina, mas elas demandam cientistas de dados treinados e uma quantidade significativa de dados de alta qualidade, o que são requisitos desafiadores para pequenas e médias empresas cumprirem. O presente Projeto de Finalização de Curso apresenta uma abordagem usando modelos híbridos, que combinam aprendizado de máquina e modelos baseados na física e possuem menos exigências em relação a dados históricos e equipe especializada. Essa solução tem como objetivo o desenvolvimento de um módulo que automaticamente cria e configura modelos híbridos para prever o desvio de forma de vidro durante o processo de conformação a quente. Para cumprir esse objetivo, foram conduzidos estudos iniciais para explorar o estado da arte das aplicações de modelos híbridos de machine learning e para entender as diferentes possibilidades de estrutura. Em seguida, os requisitos do projeto foram analizados e um fluxo de processo foi criado para descrever a geração de modelos híbridos. Posteriormente, a lógica criada foi colocada em prática utilizando Python e a validade foi verificada aplicando-a ao caso de uso.

Palavras-chave: Modelos híbridos. Aprendizado de máquina. Conformação a quente. Modelagem de vidro.

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE LISTINGS

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
AutoML	Automated Machine Learning
BB	Black-box
BO	Bayesian optimisation
CNN	Convolutional Neural Network
GA	Genetic Algorithms
GBR	Gradient Boosting Regression
IPT	Institute for Production Technology
IR	Infrared
LiDAR	Light Detection and Ranging
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NGM	Non-isothermal Glass Molding
NN	Neural Network
OOP	Object-Oriented Programming
PDE	Partial Differential Equation
PGM	Precision Glass Molding
PINN	Physics-informed Neural Network
P-V	Peak-to-valley
RMS	Root Mean Square
RMSE	Root Mean Squared Error
SMEs	Small and Medium-Enterprises
WB	White-box

CONTENTS

1 INTRODUCTION

The use of machine learning has been constantly growing over the last years, due to the numerous benefits and capabilities of these data-driven algorithms. However, there are some situations in which traditional engineering knowledge still provides valuable insights.

Hybrid models are a solution that holds the potential to revolutionize production by combining the advantages of machine learning and the usage of expert knowledge to solve complex prediction problems. The aim of this bachelor thesis is to develop a configuration framework that creates hybrid models, applied to the use case of process optimization of complex optics by predicting the shape deviation.

This initial chapter will outline the context and motivation for the work, followed by an introduction to the OptiMassKI project, in which the thesis is contained. The general and specific objectives of the bachelor thesis will then be detailed, along with a description of the scope of the work. Finally, the document structure is presented.

1.1 MOTIVATION AND CONTEXT

Over the last few years, the hot forming of glass has increased its relevance in the production process of sophisticated technical optics and thin glasses. These items are crucial to several growth markets in Germany, such as the automotive, electronics, and tech industries.

Production of complex optics through hot forming processes is very sensitive and the quality of the output product is easily influenced by variables such as temperature, that are difficult to measure (YAN et al., 2009). The prediction of aspects about the quality of the final product, such as the shape deviation, can be used to identify issues in production and improve the overall process.

Those aspects can be predicted using simulation models, expert knowledge about the process flows and their physical laws, and Machine Learning (ML) algorithms. However, expert knowledge requires a high level of expertise and may not sufficiently consider conditions such as environmental influences and machine configurations. Furthermore, the use of ML algorithms, although able to solve complex problems, requires trained data scientists and often a large amount of data, which can be a problem, especially for Small and Medium-Enterprises (SMEs) (KERSTING, 2018).

Hybrid modeling is a solution that combines both process knowledge and machine learning. The hybrid models addressed in this document consist of a white-box part (mathematical-physical model) and a black-box part (data-driven model) (RUDIN, 2019). It is intended to find out if this framework can deliver better performance, requires a smaller amount of data, and also provide explainable and comprehensible results.

1.2 OPTIMASSKI PROJECT

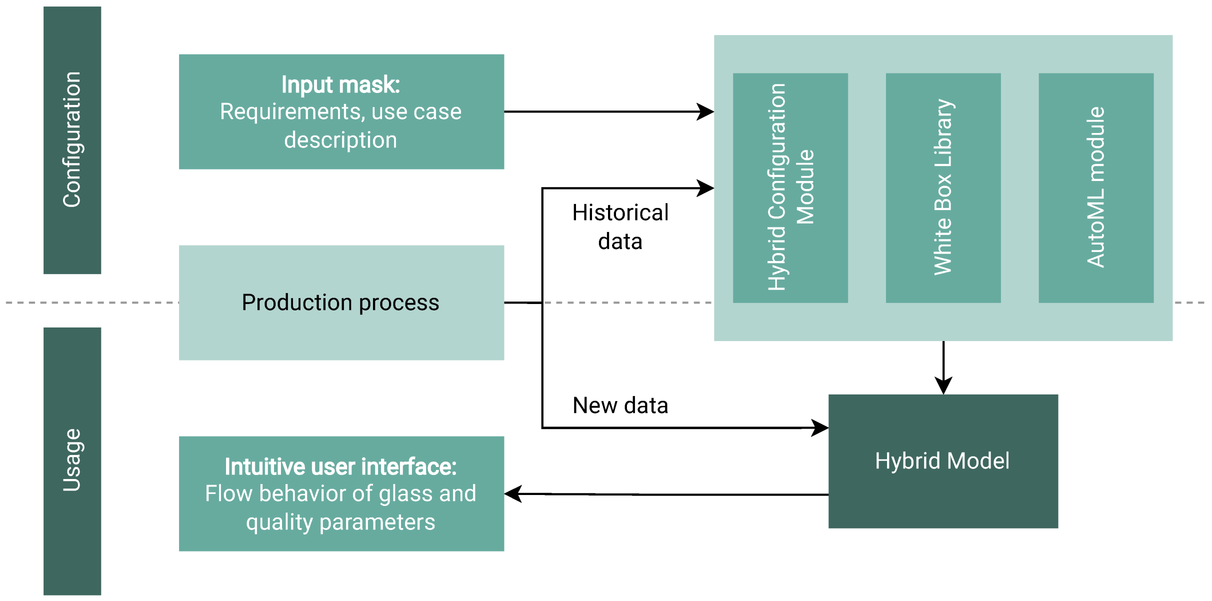
This thesis is part of **OptiMassKI**, an ongoing research project conducted by the Fraunhofer Institute for Production Technology (IPT) and supported by *Forschungsvereinigung Feinmechanik, Optik und Medizintechnik* e. V. (Research Association in Precision Mechanics, Optics and Medical Technology) or F.O.M..

With the goal of optimizing the hot forming process, the project aims to predict the flow behavior of glass through the use of hybrid machine learning. This will involve the creation of a hybrid model, in which the black-box element is trained with measured data from the process and the white-box element consists of a rheological model, expert knowledge, or simulations. Each of these features will be implemented in a user-friendly software tool that supports two phases: configuration and usage of the models.

The configuration phase consists of gathering the requirements for the result and the description of the use case, as an input mask, configuring and training the hybrid model with historical data, and making it available for usage. Then, the usage phase consists of inputting new process data into the designed hybrid model to obtain the prediction of the desired values. The final result should be easy to interpret from the user's point of view, without the need for specialized ML knowledge.

Figure 1 provides a graphical representation of the overall concept of OptiMassKI and the software tool's phases.

Figure 1 – Representation of the overall concept of the OptiMassKI project.


Configuration
Usage
Input mask:
Requirements, use case description
Production process
Intuitive user interface:
Flow behavior of glass and quality parameters
Historical data
New data
Hybrid Configuration Module
White Box Library
AutoML module
Hybrid Model

Source: Fraunhofer IPT (2022).

1.3 OBJECTIVE OF THE THESIS

1.3.1 General objective

As part of the OptiMassKI project, the bachelor thesis aims to develop a framework that automatically generates and configures a hybrid ML model, combining models from a white-box library and Automated Machine Learning (AutoML), for the black-box part. This configuration component should process admin and user inputs and return as an output a suggestion of hybrid models that should proceed in the prediction pipeline.

In the context of the bachelor thesis work, the configuration framework will be applied in the process of predicting the shape deviation in the production of complex optics. However, the framework should ideally be modular and possibly be applied in different use cases, or under different inputs and updates of the white-box library.

The challenges of this work are describing and characterizing the white-box models, choosing the structure of the model, and defining the criteria to choose the appropriate white-box and black-box models. The final result should also explain what are the user inputs required to enable automatic configuration, and how the configuration module should stand in the prediction pipeline and a user-friendly software tool.

1.3.2 Specific objectives

Based on the main objective describes, specific tasks were defined to achieve a configuration framework that satisfies all requirements:

- Study the literature in search of a basis on the structure of hybrid models and how they are usually applied.
- Creation of the concept that describes the functionalities of the configuration framework.
- Definition of a guideline on how to proceed with the training of hybrid models.
- Implementation of the configuration framework into code.

1.4 DOCUMENT OUTLINE

This document is divided into seven chapters to detail the development of the hybrid model configuration framework.

Chapter 1 explores the context and motivation for the OptiMassKI project, in which the bachelor thesis is contained. It also details the scope of the student's work.

Chapter 2 addresses the theoretical background of the thesis. Thus, it outlines concepts of glass molding and machine learning that are fundamental for understanding the remaining chapters.

Chapter 3 focuses on the presentation of the state of the art on machine learning applied to glass science and hybrid models. Therefore, it summarizes recent academic works on the topics and also presents the author's conclusions about the research.

Chapter 4 presents the concept of the developed configuration framework. It provides an overview of the project requisites and initial assumptions. Moreover, it details the configuration process, including the requirements and limitations of each step.

Chapter 5 covers the implementation of the configuration framework using Python scripts. It discusses the technologies used and describes the class structure developed to implement the configuration module. This chapter ends with a validation of the created program applied to the use case.

Finally, **chapter 6** concludes the work and discusses the result of the thesis compared to the initial objectives. It also provides an outlook on further work regarding the OptiMassKI project.

2 THEORETICAL BACKGROUND

This chapter comprehends a theoretical overview of important subjects for the thesis. In the first section, the glass molding process is outlined, highlighting the techniques that are involved in the project. The second section addresses machine learning and artificial intelligence, providing a summary of the fundamental concepts and a summary of the most used algorithms.

2.1 GLASS MOLDING

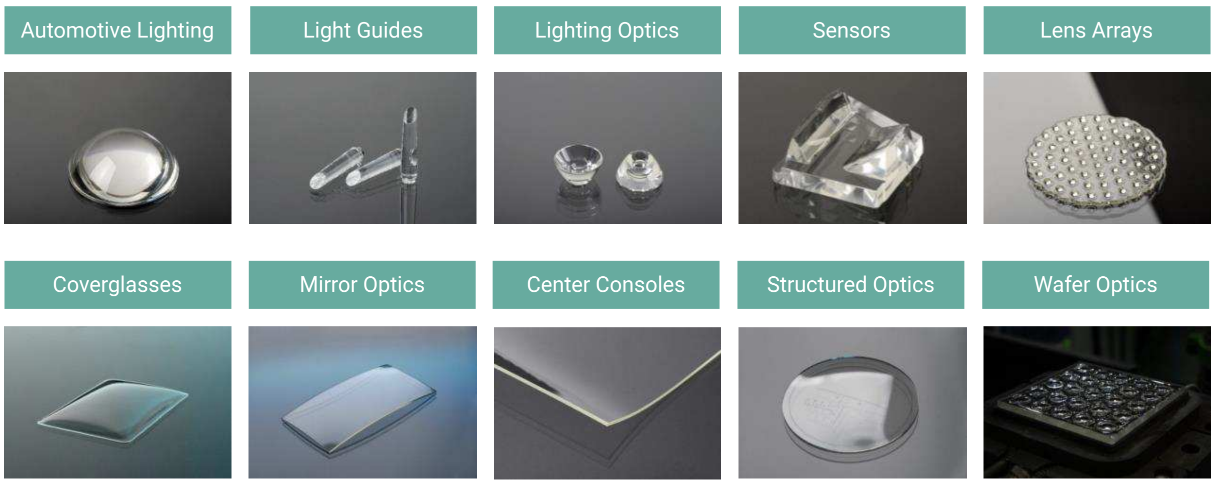
2.1.1 Overview of the production of complex optics

Recently, the automotive industry became the most relevant growth market for products made of thin glass. To meet the demands of autonomous driving and driving assistant systems, it is necessary to manufacture complex lenses and thin glasses. Examples of such optic products are the glasses used for Light Detection and Ranging (LiDAR) and laser-based systems.

The market volume of systems for driving assistance, in which LiDAR is included, is expected to triple in the next decade. (BBC, 2021) In fact, optics and photonics industries in Germany, in which hot-forming glass is applied, are large, with a turnover of 37.5 billion euros in 2019 and employment figures of over 140,000. A considerable fraction of these companies are SMEs.

Hot forming of glass is used for the production of complex optics in numerous markets. Some different products of these manufacturing processes are displayed in Figure 2.

Figure 2 – Examples of glass products manufactured with hot forming.



Source: Author.

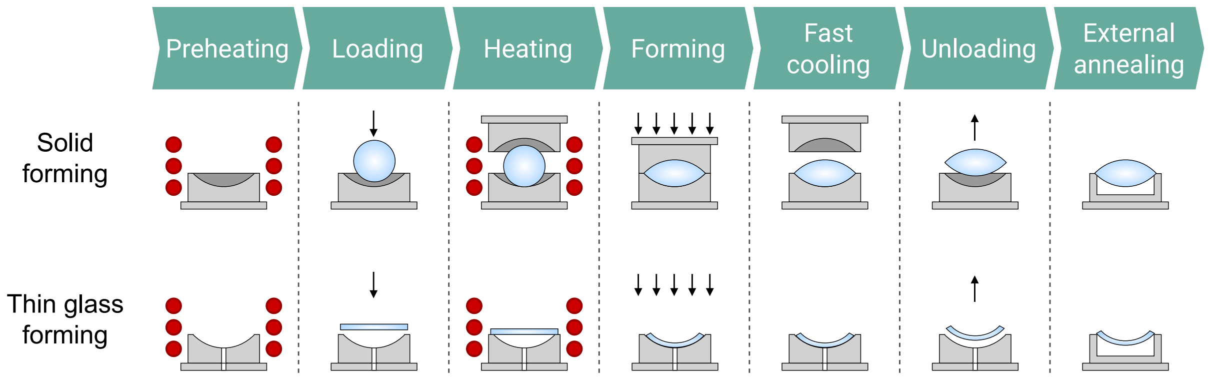
The glass molding press is considered a promising approach for the efficient production of optical elements with complex shapes. Traditionally, this process is performed using the Precision Glass Molding (PGM) method, also known as isothermal glass molding, in which the pressing is done when the temperature of the glass is the same as the temperature of the mold. However, the isothermal molding process implies a high time consumption in the heating and cooling stages. An alternative method is non-isothermal glass molding, which decreases the molding cycle time and can also extend the mold's life. (ZHOU et al., 2010; VU, A.-T. et al., 2016)

2.1.2 Non-isothermal glass molding

Non-isothermal Glass Molding (NGM) is a replicative hot-forming process based on the concept that the glass and the mold assume different temperatures. The blank pressing process is used in solid forming. The glass is heated to a value higher than its transformation temperature, while the mold press mold is separately heated to a lower temperature. Then, the mold surface is additionally heated right before the pressing process, which lasts a few seconds. Finally, the glass component is cooled in a furnace. (BLIEDTNER; GRÄFE, 2008; FELDMANN; KASPER; LANGOSCH, 2012)

For thin glasses, a vacuum-assisted process is applied. The steps are analogous to solid forming, with the difference that the driving forming forces are executed by gravity and a vacuum in the forming cavity. Both the blank pressing and the vacuum-assisted processes are represented in Figure 3. (MENDE et al., 2022)

Figure 3 – Non-isothermal glass forming process principles.



Source: Author, based on Mende et al. (2022).

Due to the heat exchanges and the highly complex material behaviors, the molding process usually causes imperfections in the produced components. In serial production, it is expected that these faults, if not detected in the molding step, lead to a considerable amount of waste and energy consumption. Root Mean Square (RMS) and Peak-to-valley (P-V) are commonly used methods to measure the surface form deviation, as they describe how well the molded shape matches the expected form. In the study conducted by Mende et al. (2022), the shape deviation is measured by collecting the value of the peak point of the lens and two equally spaced points on either side of the peak. (VU, A. T. et al., 2022)

2.2 MACHINE LEARNING

2.2.1 Fundamentals of artificial intelligence and machine learning

In order to understand the concept and state of the art of hybrid models, it is crucial to have an overview of ML, one of the branches of Artificial Intelligence (AI).

According to McCarthy (2007), founder of the field, AI is "the science and engineering of making intelligent machines, especially intelligent computer programs." The research on AI grew after the Second World War, especially with the English mathematician Alan Turing, who was the one to advocate for the shift from machine-building approaches to programming-based methodologies. By the end of the 1950s, the research in AI had many contributors, most of them having their work based on programming computers.

While the objective of AI is quite broad and includes multiple tasks involving problem-solving, reasoning, and general learning, the field of ML studies specifically the learning process (KERSTING, 2018). T.M. Mitchell (1997) defines ML as the science "concerned with the question of how to construct computer programs that automatically improve with experience".

Using statistical methods, ML algorithms are trained to perform tasks such as predicting values, classifying elements, and identifying patterns or outliers within given data. These ML methods fall into three main categories: supervised, unsupervised, and reinforcement learning. (RAVINDER et al., 2021; JORDAN; MITCHELL, T. M., 2015)

Supervised learning algorithms, the most used approach, have the objective to identify the relationship between input features and corresponding outputs, sometimes called "labels". To learn this association, the ML model is provided with "training" data, composed of a X dataset, that represents the inputs, and a y dataset to represent the outputs. This method can be used for classification or regression (prediction) tasks and some of the most common algorithms include decision trees, logistic regression, and neural networks. (JORDAN; MITCHELL, T. M., 2015)

Unsupervised learning methods are used in the analysis of unlabelled data to identify hidden patterns. Examples of applications using this approach involve clustering, which is grouping the data by similar features, and outlier detection. Some of the most common algorithms are k-means and k-Nearest Neighbours (k-NN). (JORDAN; MITCHELL, T. M., 2015)

Finally, in **reinforcement learning** algorithms, the agent learns the action that maximizes the reward. The training data in this case only offers a signal indicating if the action is correct or not, rather than specific instructions or labels. This method is commonly used in games, multi-agent systems and to solve optimization problems. (RAVINDER et al., 2021; JORDAN; MITCHELL, T. M., 2015)

2.2.2 Pipeline of a machine learning project

Developing a ML project goes beyond training the model. It is important to know the main stages of this process to understand its limitations and challenges, which are outlined subsequently. Some of the steps of the process discussed here are general, but the focus is on supervised models, which is where falls the prediction task of the bachelor thesis.

After extracting the data and executing a preliminary analysis of the information, one of the first steps is to split the data into two or three groups. The **training set** contains data that is used to train the ML algorithm. It is usually composed of 70% of the available data. The **validation set** includes data that is used to fine-tune the hyperparameters of the ML algorithm. It is usually composed of 20% of the available data. Finally, the **test set** incorporates data that is used to evaluate the performance of the model, after the training is complete. It is usually composed of 10% of the available data. In some cases, the data is only split into training and test sets, being generally composed of 80% and 20% of the data, respectively. This split is important to avoid bias in the ML algorithm. When working with separate sets, it is possible to evaluate the actual learning capacity of the model, instead of its ability to "memorize" the results, given an input. (GOODFELLOW; BENGIO; COURVILLE, 2016; GÉRON, 2019)

The next important step concerns preparing the data for the ML algorithms. This involves mostly cleaning the data and preprocessing the formats and necessary features. Although simple in concept, this is commonly one of the most time-consuming stages of the ML pipeline. (GÉRON, 2019)

After the data is adequately prepared, the next step is to select and train the model. This is hardly a straightforward step, as it usually requires multiple iterations to fine-tune the model and even change the selected algorithm, if necessary. (GÉRON, 2019)

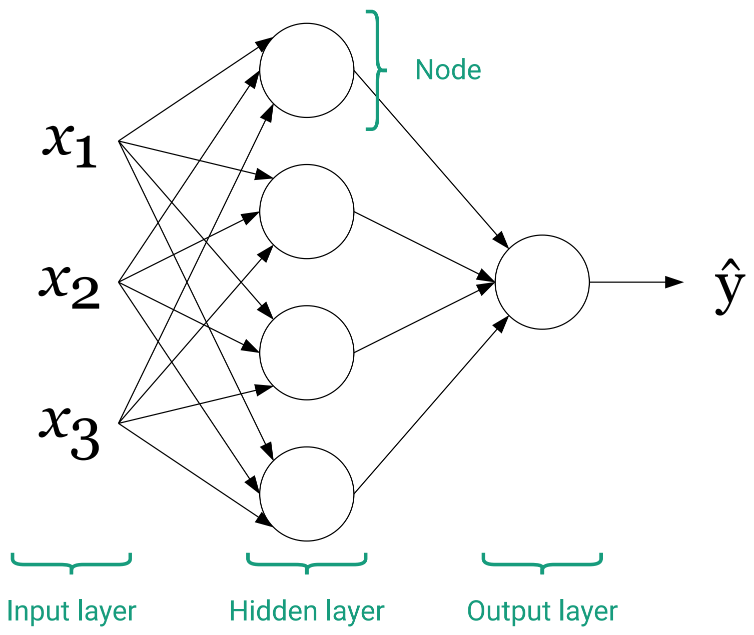
Finally, the model is evaluated. There are several ways to assess the performance of a ML algorithm, and they differ according to the objective of the model. Some of the most common evaluation metrics for regression tasks are the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE), and the Coefficient of Determination (R^2). The success of an ML model will be indicated by its ability to produce small training errors and produce test errors that have little disparity towards the training error. (GOODFELLOW; BENGIO; COURVILLE, 2016; GÉRON, 2019)

2.2.3 Machine learning algorithms

As mentioned previously, regression tasks fall under the category of supervised learning. This is the case of the ML task of this bachelor thesis project, which aims to predict the shape deviation of glass in the hot forming process.

Even inside this category, there are numerous ML algorithms to choose from. Among the popular methods are linear regression, polynomial regression, decision trees, and random forest. The most notable, or at least most famous, ML method is Artificial Neural Network (ANN), often called simply Neural Network (NN). The subfield that studies these algorithms is called deep learning. Neural networks are composed of layers, which, in turn, are composed of nodes, as shown in Figure 4. (GÉRON, 2019)

Figure 4 – Generic structure of a simple neural network.

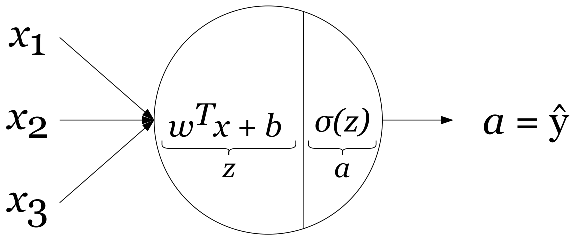


Source: Author.

Each node has its own parameters *w* and *b* and applies a linear function followed by an activation function on the input vector. In the final layer, the node's output is the prediction result. This logic is represented in Figure 5. (GÉRON, 2019)

Because of their detailed structure, neural networks are usually applied to infer complex knowledge, such as ones involving the evaluation of images in the field of computer vision. The complexity of this type of model also makes them commonly used in applications in glass science, as discussed in the next chapter.

Figure 5 – Representation of the returned output for each node in a neural network.



Source: Author.

2.2.4 Challenges of machine learning

Despite the field's immense potential and recent growth, there are several challenges encountered in the pursuit of developing robust and efficient ML solutions. Some of these obstacles are listed and outlined subsequently. Generally, they concern the characteristics of the provided data and of the structure of the ML model. (PERES et al., 2020; GÉRON, 2019)

- **Data availability and quantity.** ML solutions require a significant amount of data to perform properly. And as the complexity of the algorithm increases, so does the quantity of data needed. However, not only the quantity but the quality of the data provided to the ML model should be considered. The training data should be consistent, accurate and contain samples that cover the whole scenario in which the model is intended to perform. (PERES et al., 2020; GÉRON, 2019)
- **Complexity of the model.** There are numerous ML algorithms to choose to perform each kind of task. Some are more simple, such as linear regressors, and some are more elaborate, such as deep neural networks. Although common sense would suggest that more complex models lead to better outcomes, that is not always the case. In noisy datasets, for example, complex models would detect patterns in the noise itself, producing inaccurate results. (PERES et al., 2020)
- **Governance and explainability.** Most ML models have a "black-box" nature. That is, it is not exactly clear *how* the algorithm managed to learn the pattern from the data and produce outcomes. (PERES et al., 2020)

Those challenges are considered when assessing the performance of the ML algorithms. When the training error is large, it could represent an underfitting case. This means that the model is not being able to fit a function to the provided data. In this case, the complexity of the model should probably be increased. (GÉRON, 2019)

If the model performs well on the training set but produces significantly worse results with the test data, a case of overfitting is likely to be occurring. This could be addressed by reducing the complexity of the model or increasing the amount or quality of the data. (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.2.5 The No Free Lunch Theorem

One interesting concept proposed by Wolpert (1996) is the *No Free Lunch Theorem*, which states that there is no ML algorithm that is universally better than another. Therefore, the goal of ML is not yet to find a universal answer for all problems but to identify the tools needed to produce the best results in each case.

Since the beginning of the studies on ML much has evolved and now it is common to encounter complex models that can produce great outcomes in unexpected scenarios, sometimes even better than humans. Nonetheless, the limitations still exist and sometimes they make it unfeasible to implement ML learning solutions.

The next chapter outlines the use of ML to address problems in the context of optics and also introduces the state of the art on hybrid models, a solution that combines machine knowledge with physical principles, overcoming some of the challenges in ML that were listed previously.

3 STATE OF THE ART

The present chapter intends to provide an overview of relevant academic works that will serve as a basis for the development of this thesis.

The first section addresses applications of ML in glass science, outlining the involved challenges, and providing a summary of significant research projects in the field. Subsequently, the second section approaches hybrid models, stating a comprehensive description of the fundamentals, applications, and definition of the structure. The chapter ends with the author's conclusion about the literature survey.

3.1 APPLICATIONS OF MACHINE LEARNING IN GLASS SCIENCE

3.1.1 Challenges and considerations

Even though glasses have been part of the human routine for millennia, the optimal serial production of complex optic products is still a challenge. Glasses have a disordered nature, which makes them difficult to deal with when the objective is to measure the output quality of a moulding process.

The focus of this thesis is on the hot forming process of thin glasses. In the context of serial production, defects must be detected in the moulding phase, to avoid glass rejects and the energy cost involved in repairing product failures (VU, A. T. et al., 2022).

However, due to the glass's complex structure, such defects are difficult to measure. And modelling the values using physics-based systems should also lead to inaccurate results because of the nonlinearities existent in the process. Given this situation, ML comes as an interesting approach, with the offer to produce data-driven models that can predict glass properties in numerous applications, including, but not exclusive to, the hot forming problem. (LIU et al., 2021)

Even though ML is a promising tool in glass science, many obstacles come along. As discussed previously, ML models rely on the quantity and quality of the data to deliver a great performance. Nevertheless, it is quite challenging to build a glass dataset that is consistent, complete, accurate, and numerous. ML applications in glass science usually suffer from bad extrapolation ability, as the dataset is usually restrained to a certain range of operations and the model does not learn how to behave in extremely different environments. (RAVINDER et al., 2021)

It is also demanding to detect and deal with outliers, given that the disordered nature of glasses makes the dataset itself contain nonlinearities and anomalies. In addition, in ML it is extremely difficult to interpret the "learned" relationship between the inputs and outputs, which compromises the reliability of the model and eliminates the opportunities for new insights. (RAVINDER et al., 2021)

3.1.2 A review of machine learning in the production of optics

Within glass science and engineering, ML can be employed in numerous situations, including simulations of structural engineering applications (KRAUS; DRASS, 2020), design of glass types (LIU et al., 2021; RAVINDER et al., 2021) and quality control (LIANG et al., 2019; PARK et al., 2020).

The probable pioneer study using ML in glass science was conducted by Brauer, Rüssel, and Kraft (2007) and applied a neural network to predict the solubility of the glass P_2O_5–CaO–MgO–Na_2O–TiO_2 as a function of its composition. Many posterior works had similar approaches. (LIU et al., 2021)

A recent work by Tandia, Onbasli, and Mauro (2019) describes how Corning, one of the major companies in glass research and creator of Pyrex® glassware, used data-driven models, such as neural networks and Genetic Algorithms (GA) to enhance their products. The study outlines the development of composition-dependent models to predict key glass properties: liquidus temperature, viscosity and Young's modulus.

Their work used a neural network approach to model the liquidus temperature. The best results were found when employing Bayesian optimisation (BO) to select the best NN architecture. The same method was used to predict glass viscosity. However, it was using a Physics-informed Neural Network (PINN), combined with BO to define the best architecture, that the optimal performance was achieved. The PINN was built based on the MYEGA model for glass viscosity. A PINN model is a type of hybrid ML model and details and elaboration on this topic will be provided in a subsequent section. Finally, for predicting Young's modulus, the model's robustness is even more relevant than its accuracy. Since standard NN usually fail to extrapolate, the best-performing approach was found when employing a genetic algorithm method. In this way, it was possible to obtain great predictions for the compositions in extreme regions of data. On account of this work, it was demonstrated that ML approaches allow researchers to advance the process of creating new material by predicting glass properties. The use of data-driven methods contributed to making Corning's damage-resistant glassy products used in over 4.5 billion devices.

With respect to the hot forming of thin glass, an interesting work was conducted by Anh Tuan Vu et al. (2022). The focus of this piece was on using Infrared (IR) thermography and ML to predict the final shape of the glass after a molding process. An IR camera was placed after the molding step to collect time-series images recording the temperature fields of the glass. These images were pre-processed before they were given as inputs to the learning models.

Three approaches were assessed to apply Convolutional Neural Network (CNN) in the prediction task. The first was using transfer learning, the second used a self-built 2D CNN model and the third employed a 3D CNN model with similar architecture to the CNN-2D, except for the three-dimensional input and the use of intermediate 3D layers. This last approach was the only one that considered the time-series data, as the third dimension of the input tensor refers to the time component (frame number). The first two approaches only used the first frame of the collected data. Among the first and second approaches, the transfer learning model using DenseNet-169 delivered the best results. However, the self-built 2D CNN model was sufficiently accurate while presenting low complexity, which makes it more adequate for real-time applications, such as the one envisioned by the study. Regarding the 3D CNN model, it presented lower mean and minimum error values, compared to the 2D CNN. However, it has a high level of complexity and consequently a long training time. As a result of this work, it was presented a feasible method to employ real-time quality control in the hot-forming process using ML. This is a promising result to minimize the efforts for measuring precision glass during production.

3.1.3 Research conclusions

It is possible to encounter many results in the literature that sustain that ML can be successfully employed in glass science and engineering to predict variables that are often difficult to measure or calculate based on first-principles systems. Although the use cases are usually different than the hot forming of thin glass approached in this thesis, the existent research provides some rich insights about the use of ML to predict glass properties.

It is true that the studies present a satisfying outcome, however, these results do not come easily. In both of the works that were further outlined in the previous subsection, the ML models employed were complex and often required further resources, such as the use of Bayesian optimisation to find the optimal NN architecture and the configuration of a PINN model, in the piece by Tandia, Onbasli, and Mauro (2019).

This reinforces the existence of obstacles when applying ML when predicting glass properties, as detailed previously. In relation to SMEs, the implementation of ML models becomes even more challenging, since they usually require the expertise of a data scientist and the algorithms tend not to be comprehensible. In addition, the use of additional equipment such as the IR camera employed by Anh Tuan Vu et al. (2022) is often not feasible in economic and logistic terms.

One approach that was mentioned in various reviews about ML and AI in glass science is the use of hybrid ML models. This method can overcome some of the constraints accompanying the standard ML models like the demand for large amounts of data and the poor interpretability.

3.2 HYBRID MACHINE LEARNING

3.2.1 Fundamentals of hybrid models

Hybrid models are defined as ones that combine parametric and nonparametric structures. In parametric models, the structure is fixed according to a priori knowledge, such as mechanistic and phenomenological models, (e.g., a rheological model, a simulation, or expert knowledge). On the other hand, in nonparametric models, the structure is inferred from data, which is exactly the case of ML models. Therefore, the combination of both structures results in semiparametric hybrid models, which is the focus of the bachelor thesis work. (GLASSEY; STOSCH, 2018)

By using a semiparametric structure, it is possible to obtain key advantages over both types of models, as follows:

- Increase of the model's physical significance, which also enhances its interpretability;
- Fewer requirements on the data, regarding quantity and quality and improved handling of uneven datasets, given that the parametric model outlines a structure to the system;
- Advanced extrapolation capabilities, allowing the model to act beyond the data range. (GLASSEY; STOSCH, 2018; ZENDEHBOUDI; REZAEI; LOHI, 2018)

Hybrid models were introduced around 1992 as a way to obtain more robust and interpretable neural network models. Then, Thompson and Kramer (1994) portrayed hybrid models as the combination of parametric and nonparametric models described previously. (GLASSEY; STOSCH, 2018)

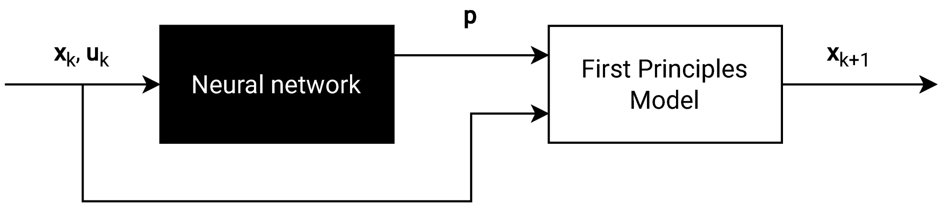
The research on the topic has been evolving ever since and many applications have been proposed, some of which will be discussed in the next section. The motivation for analysing these studies in this bachelor thesis is to explore how different types of hybrid models are employed and elucidate their architectural characteristics.

3.2.2 Hybrid model applications

The concept of hybrid modelling was introduced by Psichogios and Ungar (1992) to determine the kinetics of fedbatch bioreactors, which are difficult to model because of their dynamic characteristic. To explore the effects of including first-principles knowledge in neural networks, the study compares a standard NN model to a hybrid model approach, both applied to the same context.

The structure of the hybrid model developed by the authors is represented in Figure 6.

Figure 6 – Structure of the hybrid model used to determine the kinetics in fedbatch bioreactors.

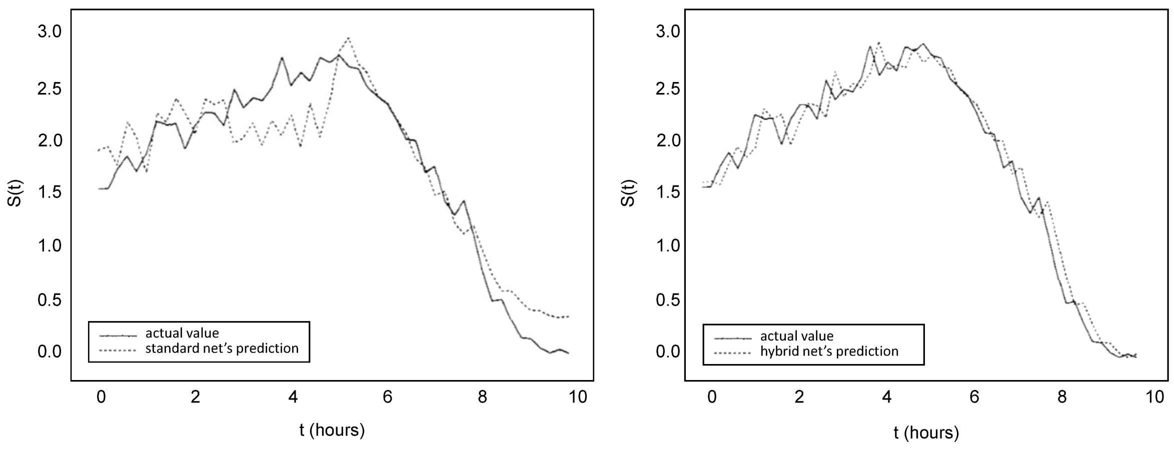


Source: Adapted from Psichogios and Ungar (1992).

The BB component receives the initial values of process variables as inputs and outputs the value of the cell growth rate. This value along with the initial process variables is then given as an input to the WB model, which returns the values of the process variables for the next sampling instant. In other words, the BB model is used to predict an unknown parameter (the cell growth rate), which is then input into the WB model to generate the final hybrid model result.

As of the standard NN approach, the BB model receives the initial process variables as inputs and predicts these values for the next sampling instant. By training both models in different scenarios according to the size of the training set, it was demonstrated that the mean squared error of the hybrid model is significantly lower (by an order of magnitude) than the error obtained with the standard NN model. The enhanced performance of the hybrid model is also exemplified in Figure 7, which shows the prediction values of both models, compared to the actual values.

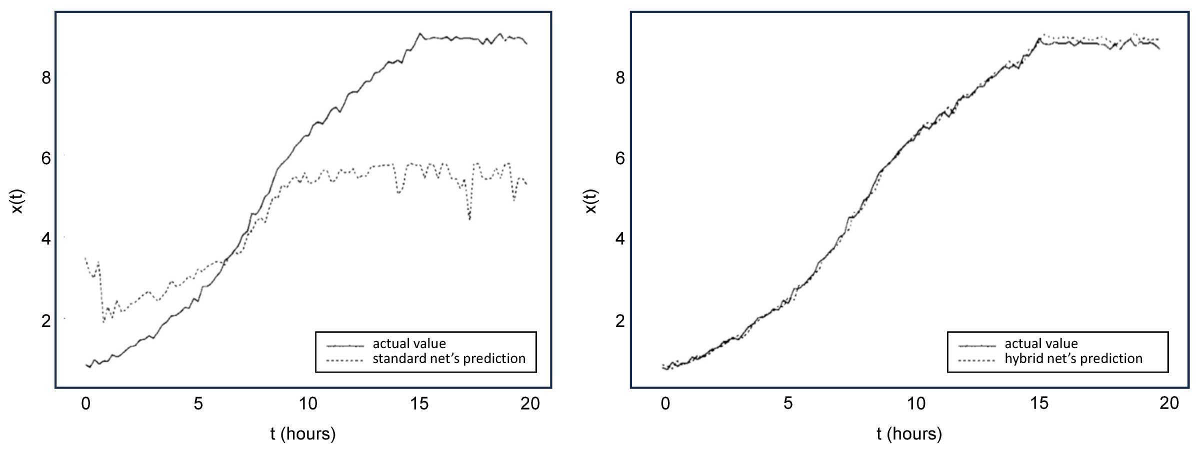
Figure 7 – Prediction results of the substrate concentration (S) over time for the standard and the hybrid models.



Source: Adapted from Psichogios and Ungar (1992).

Another expected advantage of the hybrid model is its extrapolation capability. This characteristic was also demonstrated by the study as shown in Figure 8, which plots the prediction results of both models when operated in a different state space regime than the ones sampled by the training set.

Figure 8 – Prediction results of the biomass concentration (X) over time for the standard and the hybrid models.

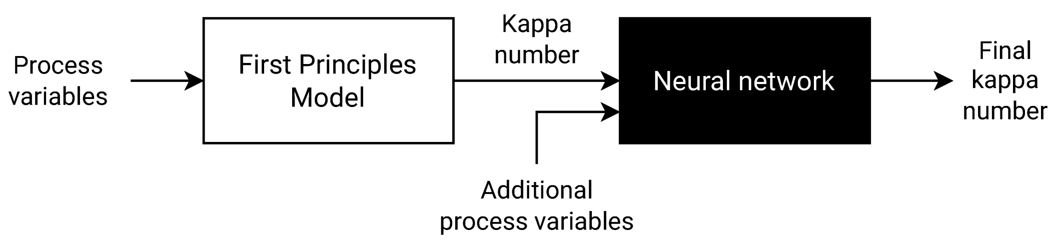


Source: Adapted from Psichogios and Ungar (1992).

Aside from being one of the pioneer studies in hybrid modelling, this study is very successful in demonstrating the enhanced performance of a semiparametric model when compared to a standard nonparametric neural network approach.

Another interesting work was conducted by Aguiar and Filho (2001). A hybrid model was used to predict the kappa number in a pulp mill, which can be used to improve pulp quality. Figure 9 shows the structure of the hybrid model that was developed in the study.

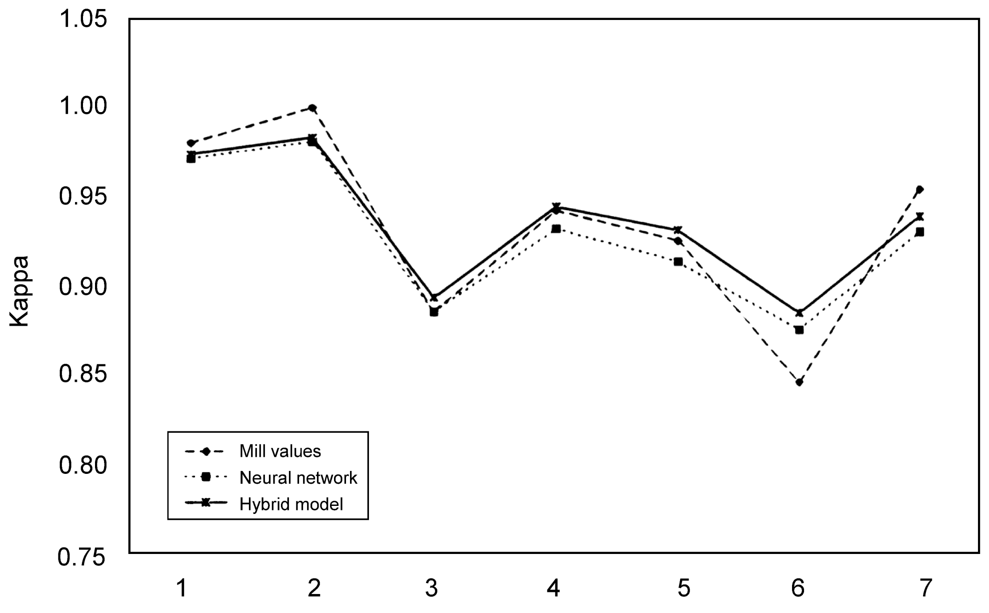
Figure 9 – Structure of the hybrid model used to model the kappa number.



Source: Adapted from Aguiar and Filho (2001).

In this configuration, the WB model uses the process variables to generate the kappa number, following kinetics equations. The output of this model is then added to other process variables and fed into a neural network, which returns the final kappa number. It is possible to evaluate the results obtained by the hybrid model, compared to a standard NN in Figure 10.

Figure 10 – Prediction results of the kappa number for the standard and hybrid models.



Source: Adapted from Aguiar and Filho (2001).

The graph demonstrates a slight improvement in the performance of the hybrid model, compared to the standard NN. However, the more drastic difference concerns computational performance, given that the hybrid model's training time was half as long as the standard net's.

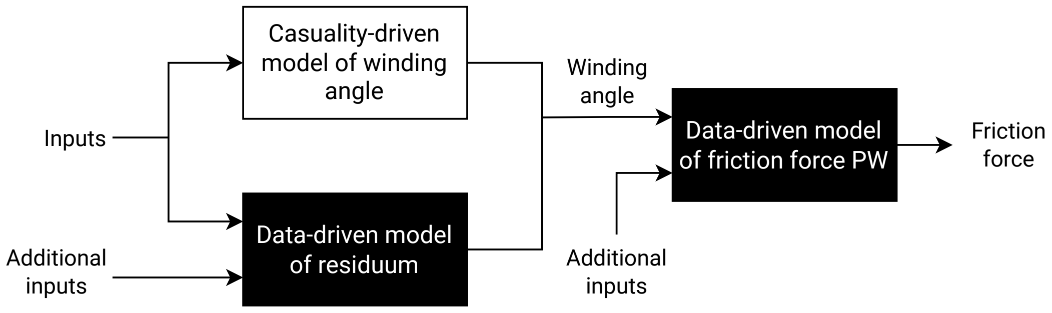
The authors point out that these results should be considered a starting point for developing more complex hybrid models.

In a study performed at Fraunhofer IPT by Dorißen, Heymann, and Schmitt (2022), a hybrid model is proposed to predict the friction forces that happen in a micro-pullwinding process, so that the operation could be intervened in order to avoid them. The combination strategy employed in the model is shown in Figure 11.

In the implemented architecture, the parametric model is combined with a non-parametric model, that predicts the residuum, in a parallel approach to determine the winding angle. This result is then added to other measured data and input into a final nonparametric model, that yields the friction force.

The study evaluates the performance of the hybrid model for the prediction of the winding angle. Compared to approaches with only the deterministic model or only a standard NN, the hybrid model using a Gradient Boosting Regression (GBR) demon-

Figure 11 – Structure of the hybrid model employed to predict friction forces in a micro-pullwinding process.


Casuality-driven model of winding angle
Inputs
Data-driven model of residuum
Additional inputs
Winding angle
Data-driven model of friction force PW
Additional inputs
Friction force

Source: Adapted from Dorißen, Heymann, and Schmitt (2022).

strated better results. The hybrid method performed better in 90% of the experiments and was able to achieve a R^2 of 0.9576.

These results prove that hybrid ML is promising and can be successfully applied in a production environment. The authors cite as future challenges the creation of an efficient approach during development to decide which model could acquire better outcomes.

In the literature on hybrid models, it is common to find approaches using **Physics-informed Neural Networks (PINN)**. This concept, introduced by Raissi, Perdikaris, and Karniadakis (2019), consists of a method to incorporate parametric knowledge into an initially nonparametric neural network.

The principle behind standard neural networks is generally based on using back-propagation to minimise the Mean Squared Error (MSE) between the predicted and the actual data points. In PINN, the NN's loss function is augmented with a term that corresponds to the physical model of the problem. This allows the NN to integrate structure and previous knowledge regarding the analysed model, enhancing its generalisation interpretation ability.

However, although the results achieved by PINN models are considered satisfactory and promising, the implementation of this method is not simple and is very case-specific. It is necessary to develop a mathematical expression that represents a physical constraint to be minimised in the form of a Partial Differential Equation (PDE), which will then be added to the neural network's loss function. (CAI et al., 2021)

In conclusion, PINN is a very interesting and relevant concept in the subject of hybrid modelling. It was proven to deliver promising predicting results and excellent extrapolation capability (VU, A. T. et al., 2021; TANDIA; ONBASLI; MAURO, 2019; WU et al., 2022). Nevertheless, this technique is not suitable for the use case approached in this bachelor thesis.

The project's main objective is to create a framework that automatically configures hybrid models and the specificity required in PINN makes this goal difficult to achieve. Furthermore, PINN involves a mathematical model founded on PDEs. That will not always be the case for the WB models of the use case, which assume different formats, being mostly represented by simulation models.

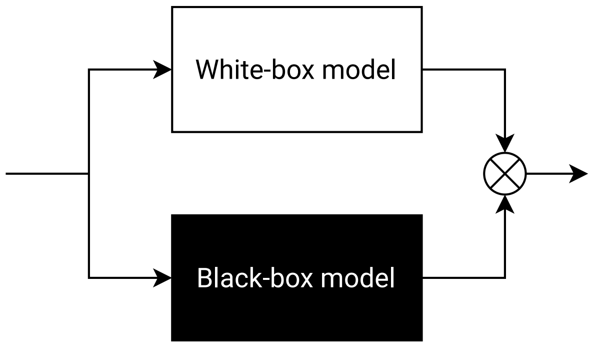
3.2.3 Structures of hybrid models

In the applications outlined in the previous section, it should be observed that different combinations of nonparametric and parametric models are employed. Hybrid models can vary in quantity and type of sub-model and even the simplest model, composed of a WB model and a BB model, can be arranged in two modes: parallel or serial. (ZENDEHBOUDI; REZAEI; LOHI, 2018)

3.2.3.1 Parallel structure

Figure 12 illustrates a semiparametric model combined using a parallel structure.

Figure 12 – Generic parallel structure of a hybrid model.



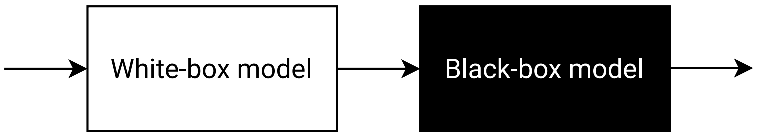
Source: Adapted from Glassey and Stosch (2018).

This type of combination is usually implemented when a full parametric model of the system is available, although it fails to provide sufficiently accurate results. In this case, a nonparametric model is added in parallel to estimate the error of the mechanistic model. The outputs of both models are then summed to form the final result of the hybrid model. (GLASSEY; STOSCH, 2018; ZENDEHBOUDI; REZAEI; LOHI, 2018)

3.2.3.2 Serial structure

Serial structures, represented in Figures 13 and 14 are employed when there are unknown parts of the mechanistic model. The nonparametric model is responsible for describing the unidentified information. A hybrid model formed by a WB model followed by a BB model is shown in Figure 13. This configuration is similar to the parallel structure, given that the BB model is used to correct mispredictions, based on the input given by the WB model. (GLASSEY; STOSCH, 2018)

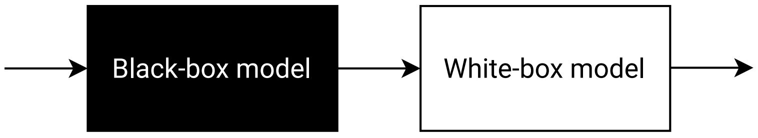
Figure 13 – Generic serial structure of a hybrid model that combines a WB model followed by a BB model.



Source: Adapted from Glassey and Stosch (2018).

The other arrangement possibility of a serial hybrid model is represented in Figure 14. This formatting is valid when detailed information about the mechanistic process is not provided. The BB model is used to predict this knowledge, based on the process inputs, and propagates it to the WB model, which computes the final result. (GLASSEY; STOSCH, 2018; ZENDEHBOUDI; REZAEI; LOHI, 2018)

Figure 14 – Generic serial structure of a hybrid model that combines a BB model followed by a WB model.



Source: Adapted from Glassey and Stosch (2018).

When it is feasible to use more sub-models, it is also possible to implement a mixed structure, that combines serial and parallel configurations (GLASSEY; STOSCH, 2018). This architecture was employed by Dorißen, Heymann, and Schmitt (2022), as mentioned in the previous section.

Finally, it is important to state that there is not a generally preferred structure. The choice of the combination method strongly depends on the available knowledge and also on the quality of the information acquired by each sub-model. (GLASSEY; STOSCH, 2018)

3.2.4 Research conclusions

The studies analyzed in this section proved that the expected benefits of hybrid models are achieved in practice. All results demonstrated that semiparametric models performed better while requiring fewer training resources (e.g. computational power and quality and amount of data) when compared to standard ML models.

The selection of the appropriate structure for hybrid models remains a challenge, given that there is no universally outperforming architecture and the choice of the preferred arrangement is strongly dependent on the available information in each use case. In addition, extensive exploration of the literature has revealed an absence of information regarding the application of hybrid models in the specific use case of the bachelor thesis project: predicting the shape deviation of glass in the hot forming process, which makes it even more challenging to estimate the ideal hybrid model structure for this prediction task.

However, research has provided a comprehensive overview of the applications of different semiparametric combinations and this knowledge will be a resourceful input to the practical work of this bachelor thesis. Furthermore, the general approach to implementing hybrid models was shown to involve three key steps: determining the structure, training the BB model, and analyzing the obtained results.

All of the references and knowledge acquired thus far will be taken into account in the upcoming chapter, which describes the development of a framework that automatically configures hybrid models for predicting shape deviation in the hot forming of glass.

4 CONCEPT OF THE CONFIGURATION FRAMEWORK

This chapter introduces the developed concept of the configuration framework. It begins by highlighting the project requisites and describing some initial assumptions. Thereafter, the configuration workflow is proposed and all of its steps are thoroughly detailed. Finally, a description of the process for training hybrid models is presented, providing a guideline for future works.

4.1 PROJECT REQUISITES

As a general objective, the OptiMassKI project aims to use hybrid models to predict the glass flow behavior and achieve process optimization in hot forming. The creation and use of the hybrid model will be automatically implemented in a user-friendly software tool, which will support mostly SMEs. Thus, the companies will be able to use explainable ML without the need for specialized personnel.

This bachelor's thesis is focused on developing the configuration module of the software tool, with the primary purpose of **automatically creating the architecture of a hybrid model**. To sustain this feature, the module should receive input from the user, describing the use case and the available process data. It should also access a White-Box library, containing options of parametric models to form the hybrid structure.

The outcome of the configuration module is crucial to the elaboration of future tasks in the OptiMassKI project including the training of the hybrid models using production data and the development of a user interface to sustain the framework. Therefore, to ensure that the configuration module meets the project necessities, some requirements should be satisfied, as listed in the following.

- The White-Box Library should be able to be populated by use-case specialists.
- The elements of the White-Box Library should sufficiently characterize parametric models.
- The user of the framework should be capable of describing the available production data and selecting the adequate WB model from the library.
- The hybrid model should have its structure configured automatically, i.e. without the intervention of the user.
- The hybrid model structure should contain a characterization of the required sub-models (WB and BB), including a description of the inputs and outputs.
- Although the use case should be used as a guideline for the development, the configuration module should be adaptable to different applications.

4.2 INITIAL CONSIDERATIONS ABOUT THE HYBRID MODEL

Prior to the development of the configuration module, some initial definitions were made concerning the available WB models and ideas of possible hybrid model architectures.

There are three types of WB sub-models considered by the use case:

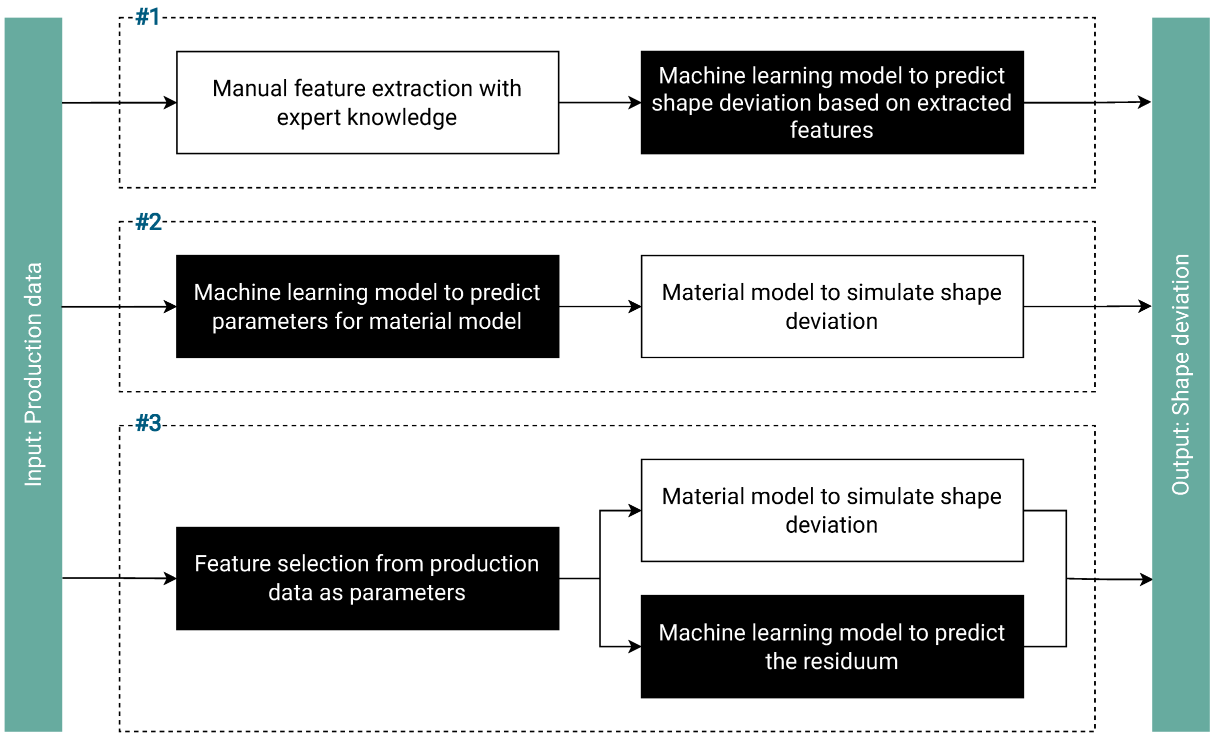
(i) a simulation model;

(ii) a rheological model (e.g., Maxwell or Burgers model); and

(iii) expert knowledge in the form of physics-informed feature selection.

In addition, regarding the hybrid model, three structures were estimated as possibilities, as shown in Figure 15.

Figure 15 – Estimate of possible hybrid model structures to be applied in the use case.


Input: Production data
#1
Manual feature extraction with expert knowledge
Machine learning model to predict shape deviation based on extracted features
#2
Machine learning model to predict parameters for material model
Material model to simulate shape deviation
#3
Feature selection from production data as parameters
Material model to simulate shape deviation
Machine learning model to predict the residuum
Output: Shape deviation

Source: Author.

The first setup concerns a serial configuration, of a WB model followed by a BB model. In this case, the WB model would be of type (iii), using expert knowledge to perform feature extraction. Then, the BB model is used to predict the shape deviation based on the selected features.

In the second structure, a serial configuration is also employed, but with a BB model followed by a WB model. The BB model has the objective to predict unknown parameters and provide them to the WB model of type (i), which uses a material model to simulate shape deviation.

Finally, the third option uses a combination of serial and parallel configurations. A BB model is initially employed with the function of performing a feature extraction of the prediction data. This is given as input, in parallel to the WB model of type (i) and a second BB model. The WB simulates the shape deviation and the final BB model predicts the residuum. Both outputs are summed to achieve the final value for the shape deviation.

These structures are merely a starting point for the development of the configuration module. This means that the options for the hybrid model structure should include but not be restricted to them.

4.3 CONFIGURATION WORKFLOW

As stated in the previous chapter, the selection of the best hybrid model structure strongly depends on the knowledge available (GLASSEY; STOSCH, 2018). Thus, it is challenging to generically select a hybrid model structure that could work in various scenarios, as it is required by the use case.

The chosen approach is to analyze the use case based on the description of the production data that is available to be input into the hybrid model and the information that is required by the corresponding WB model. With this information, it is possible to generate possibilities of hybrid model structures, that, in theory, will result in satisfactory performance. Then, these options are trained with historical production data and, based on the evaluation and comparison of each model's performance, the best hybrid structure is chosen.

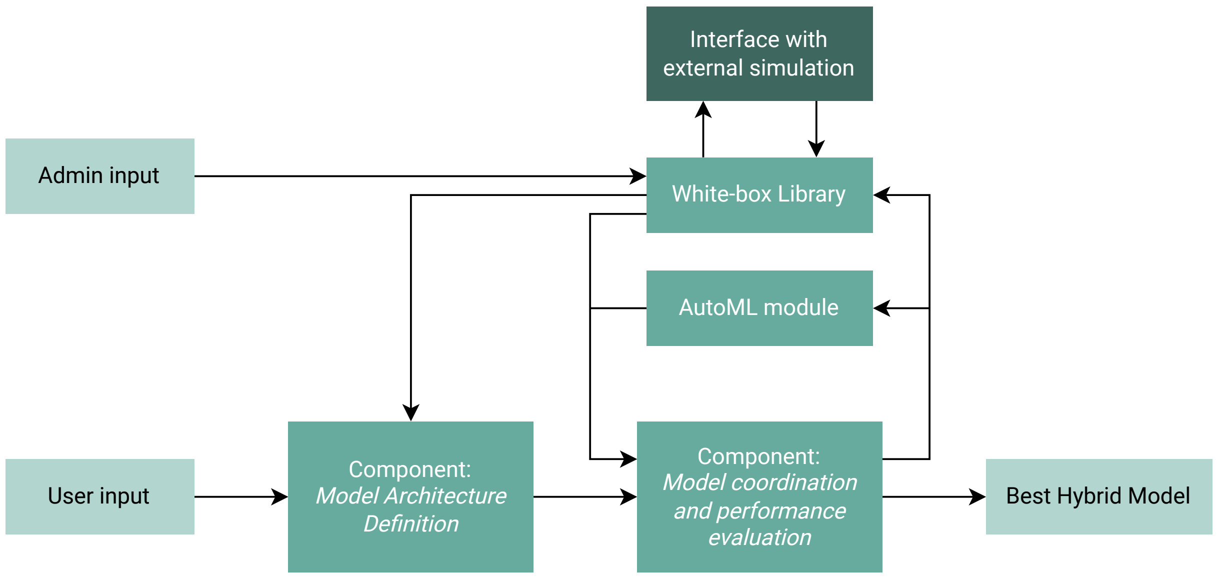
A workflow for the configuration of the hybrid model was designed, as shown in Figure 16. The diagram illustrates the relationship between the different components of the process.

There are two major components in the configuration process of a hybrid model. The first is assigned with the definition of the model architecture, based on the use case requirements given by the user. This step has three primary objectives: definition of promising model configurations, selection of an adequate WB model, and definition of the requirements for the ML model.

Then, the acquired model configuration is passed into the next component, responsible for the model coordination and performance evaluation. The general idea of this step is to train the hybrid model, especially the BB part, and evaluate the performance of hybrid models to choose the best one. The training is executed in iterations of communications with an AutoML[1] tool and the White-Box library, which should be

[1] AutoML refers to the use of algorithms to automate the process of building and optimizing ML models.

Figure 16 – Workflow that represents the process of configuring the hybrid model.


Interface with external simulation
Admin input
White-box Library
AutoML module
User input
Component: *Model Architecture Definition*
Component: *Model coordination and performance evaluation*
Best Hybrid Model

Source: Author.

filled with the existing WB models. This training process includes the execution of the WB model, which, depending on the case, might require an interface with an external simulation.

The training task has a complexity of its own. Due to this reason and the existing lack of production data, the execution of this step is not included in the scope of the bachelor thesis.

To better comprehend the configuration process, it can be divided into minor steps. These sub-tasks are outlined subsequently.

4.3.1 Loading the White-Box Library

Chronologically, the first step in the configuration of the hybrid model is to add entries to the White-Box Library. This can be considered as a setup task. In the workflow, this is mentioned as the "admin input" and should be executed by the use-case specialist, e.g., someone who fully understands the physical behavior of the production process.

When filling the library, the admin should register the existing WB models and their features. To properly characterize the model, the information provided should contain:

- The name of the model.
- The type of the model (external simulation, mathematical model, or expert knowledge).

- The core of the model. In the case of a mathematical model, this would consist of a set of equations, for example.
- The required parameters or variables to execute the model (inputs).
- The output parameters.

4.3.2 Generation of hybrid model layouts

With the setup complete and once the user inputs the use case description and requirements, this information is provided to the component that defines the model architecture. The first task within this step is to generate the potential hybrid model layouts.

To simplify this goal and to make sure that all the possibilities are explored, this step is executed without yet considering the use case limitations and requirements. In addition, some assumptions are made:

- Only one WB model is being used.
- More than one BB model can be used.
- Two BB models should not be placed consecutively in a serial configuration.
- A parallel combination can only be made with a WB model and a BB model, not with two models of the same type.

Taking these premises into account, the possibilities for hybrid structures are generated, producing different combinations of serial, parallel and mixed arrangements.

4.3.3 Evaluation of the hybrid model layouts

After the creation of all possible layouts, they should be evaluated and compared to the requisites specified by the user. It was previously stated that the structure of the hybrid model depends on the available knowledge. Therefore, the evaluation addresses two main questions involving this matter:

(i) Are all the required parameters for the WB model available as process data?

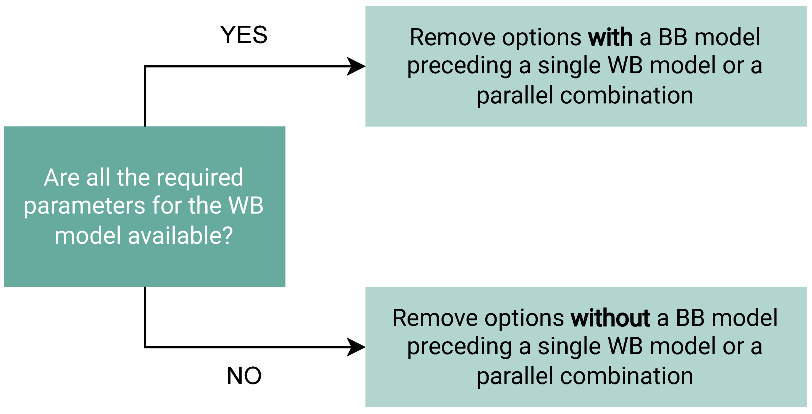
(ii) Is the output of the WB model the same as the user-required output for the entire hybrid model?

These questions should be applied to each structure generated in the previous step. As a result of this analysis, it is possible to filter the options in a way that only the feasible layouts remain in the configuration workflow. It is also then possible to characterize the sub-models concerning their objective, inputs, and outputs, which will be covered separately in the next subsection.

Regarding the first question, if all the required parameters for the WB model are available as input data to the hybrid model, there is no need to use a BB model to predict any missing information for the WB model. Therefore, the hybrid structure should begin with a single WB model or a parallel component.

Alternatively, if the required parameters for the WB model cannot be fully provided by the hybrid model's input, a BB model should be placed in front of the WB model to predict the missing information. The diagram in Figure 17 digests this logic.

Figure 17 – Evaluation of parameter availability in the configuration workflow.


YES
Remove options **with** a BB model preceding a single WB model or a parallel combination
Are all the required parameters for the WB model available?
Remove options **without** a BB model preceding a single WB model or a parallel combination
NO

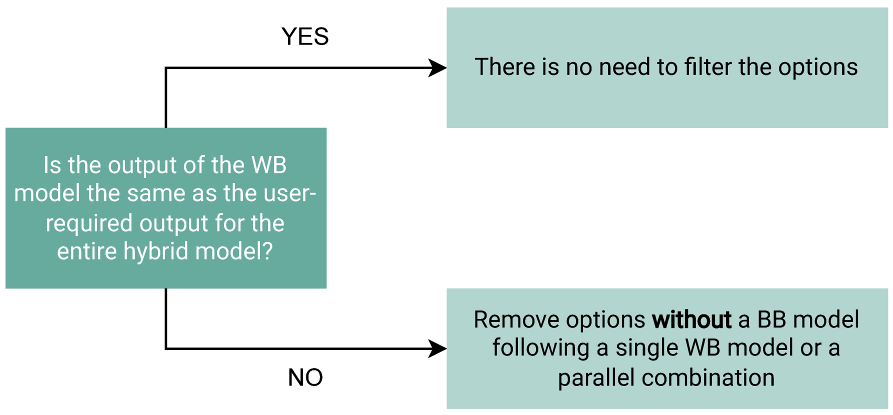
Source: Author.

As for the second question, if the output of the selected WB model is the same as the one required by the user, there is no need to remove any layout from the possibilities. It should be noted that in the options that contain a BB model following a single WB or parallel component, this BB sub-model should process the previous output to yield the final result predicting any existing noise.

The diagram in Figure 18 displays the case in which the WB model does not provide the same output as required by the user. In this situation, there must be a BB model after the WB model, or parallel combination, to predict the final result. Therefore, the options that contain a single WB or a parallel component that does not precede a BB model should be removed.

In order to demonstrate the presented logic, the diagram in Figure 19 illustrates the evaluation process of an initial layout set. In this example, it is considered that not all the required parameters of the WB model are available, therefore, a BB is required to be placed before the WB model to predict the missing information. It is also analyzed that the output of the WB model is not the same as the user-required output for the hybrid model, which demands a BB model after the WB model to predict the final result. The inspection of the first question eliminates some possibilities and after the evaluation

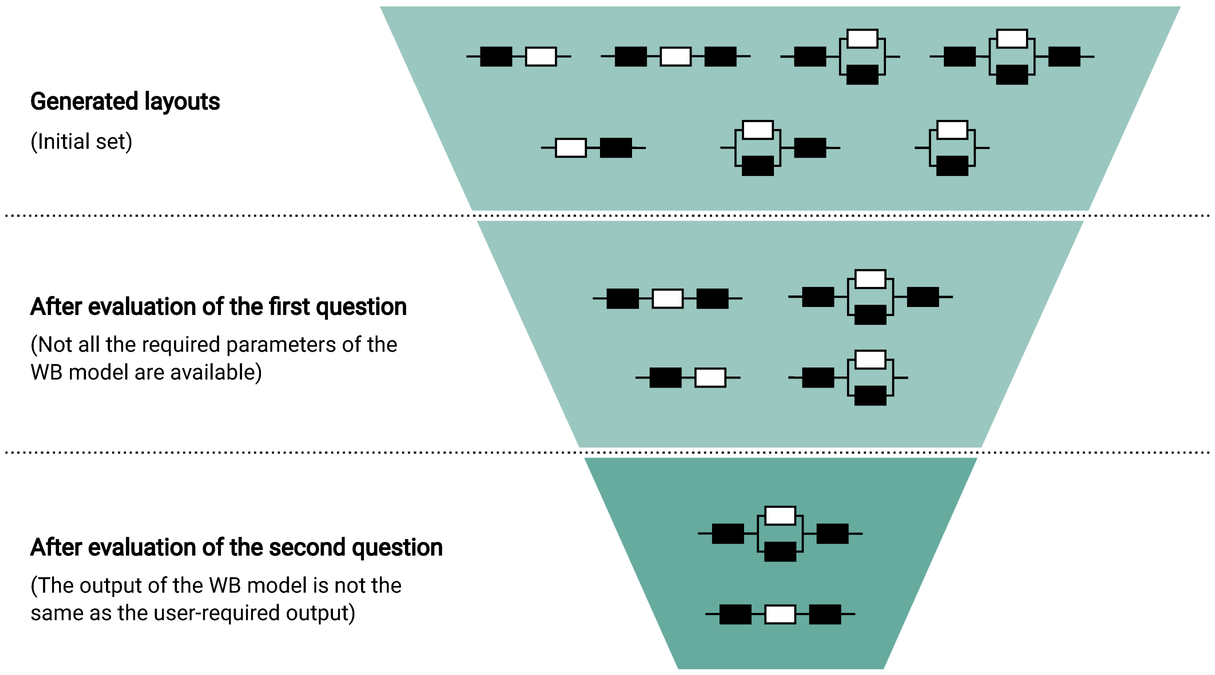
Figure 18 – Evaluation of the WB model output in the configuration workflow.


YES
There is no need to filter the options
Is the output of the WB model the same as the user-required output for the entire hybrid model?
Remove options **without** a BB model following a single WB model or a parallel combination
NO

Source: Author.

of the second question, the final layout set is defined, with the options that are coherent to the given context.

Figure 19 – Example of the evaluation process of generated hybrid model layouts.


Generated layouts
(Initial set)
After evaluation of the first question
(Not all the required parameters of the WB model are available)
After evaluation of the second question
(The output of the WB model is not the same as the user-required output)

Source: Author.

4.3.4 Characterization of the sub-models

To fully define the hybrid model architecture, the remaining task is to, for every layout option, describe and specify each sub-model by detailing its objective, inputs, and outputs. The sub-models can be specified by their occurrence in the layout, as outlined subsequently.

In the case of the **white-box models**, their inputs and outputs are predetermined by their characterization in the WB Library. Therefore, this depiction will be maintained in this step.

Regarding the **black-box models placed in parallel with a WB model**, there is only one possible objective. Their purpose is to predict the irregularities in the result produced by the WB model in parallel. This is achieved by receiving the same inputs as the WB model and generating the corresponding noise as output. An example of this depiction is the first BB model of the structure presented by Dorißen, Heymann, and Schmitt (2022).

To characterize the single BB models, that are placed in series with other components, it is crucial to look into the analysis of the questions made in the previous subsection.

When not all the required parameters for the WB model are available and a **black-box model is placed preceding a WB model or a parallel configuration**, its function is to predict the unknown information. To achieve this, it receives the same inputs as the hybrid model and outputs the missing parameters. This logic is employed in the model suggested by Psichogios and Ungar (1992).

Concerning the situations with a **black-box model following a WB model or a parallel configuration** it receives as input a combination of the hybrid model input and the output of the previous component (a single WB model or the parallel configuration), and it generates the desired output for the hybrid model. However, the objective of this BB model depends on the output of the WB model.

If the output of the WB model is the same as the hybrid model's, then the BB model's function is to predict the noise. An example of this scenario is found in the architecture proposed by Aguiar and Filho (2001). On the contrary, the BB model has the purpose of predicting the desired output of the hybrid model, as is the case of the second BB model in the structure presented by Dorißen, Heymann, and Schmitt (2022).

4.3.5 Description of the training process

Although the scope of this thesis does not encompass the implementation of the training of the hybrid model, it is important to outline how this step should proceed. The training task requires understanding the concept of the hybrid models and the generated architectures. Thus, a description of this process is crucial for the continuity of the work and to guarantee that it is feasible.

It is necessary to train the hybrid models because they are composed of BB models, which require training. In this step, the WB models only need to be executed in order to obtain the required data to train the BB models.

This step involves feeding the necessary historical data to the sub-models, contrary to the model definition step, which only demanded a description of the data. For the WB model, it is only required to provide the input data, while it is needed to supply input and output data to train the BB models.

Prior to describing the general training process, it is crucial to comprehend the execution of the WB model. The steps of this task vary according to the type of the model, as detailed subsequently.

- **Automatic external simulation**
  1. Set the values of the required parameters.
  2. Trigger the execution of the simulation program.
  3. Set the value of the output parameters based on the result of the simulation.
- **Manual external simulation**
  1. Set and export the values of the required parameters.
  2. Save current information about the training process.
  3. Pause execution and wait for user to continue with execution.
  4. Resume execution.
  5. Set the value of the output parameters based on the result of the simulation.
- **Mathematical model**
  1. Set the values of the required parameters.
  2. Evaluate the equations that characterize the WB models.
  3. Set the value of the output parameters based on the result of the simulation.
- **Expert knowledge**
  1. Get the input parameters.
  2. Wait for the user to perform a manual feature selection based on physics knowledge.
  3. Set the selected parameters as outputs.

The training process is focused on training the BB sub-models of the hybrid structure. In future works, an AutoML tool is going to be used to execute this task. Therefore, the only requirement, at the moment, is to determine or provide the necessary training data.

Generally, the BB model can be placed in three different ways: in parallel with a WB model, following a WB model/parallel configuration, or preceding a WB model/parallel configuration.

When the BB model is placed in parallel with the WB model, the output dataset is not directly available to train the BB model. Hence, the WB model should be executed and the output dataset is then obtained by calculating the difference between the global and the WB model outputs.

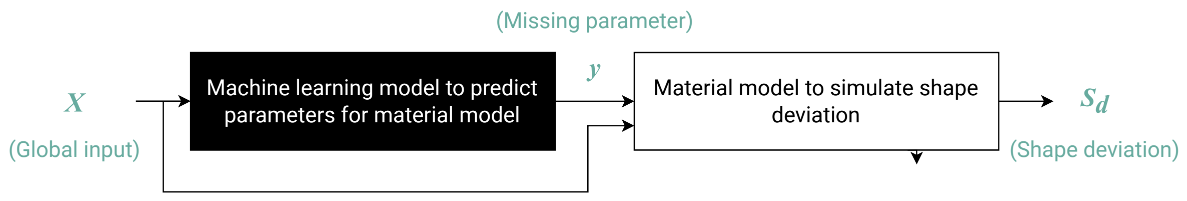
In the case of the BB model being placed following a WB model, the input dataset is not directly available to train the BB model. Thus, the WB model should be executed to acquire this data. This is analogous to the procedure when the BB model is placed following a parallel component.

The situation in which the BB model is placed preceding a WB model or parallel configuration is not as simple to handle. In this case, the BB model is used to predict the missing parameters of the WB model. However, given that this is unknown information, there might be cases where there is also no historical data to train the BB model.

Two approaches are proposed to handle this case by generating a training dataset of unknown information. The first involves creating a ML model to identify the relation between the missing parameters and the global input and output, with the help of the WB model. The second option is to use an optimization algorithm to find values for the missing parameters that, when applied to the WB model, produce the closest value to the known global output.

These approaches can be exemplified using the second hybrid model structure presented in section 4.2, as shown in Figure 20.

Figure 20 – Example of a hybrid model structure where the missing parameters are not available as historical data.


(Missing parameter)
X
(Global input)
Machine learning model to predict parameters for material model
y
Material model to simulate shape deviation
S_d
(Shape deviation)

Source: Author.

In this case, the first approach consists of the following steps:

1. Run the simulation n times using (different) values of X and y to obtain S_d.

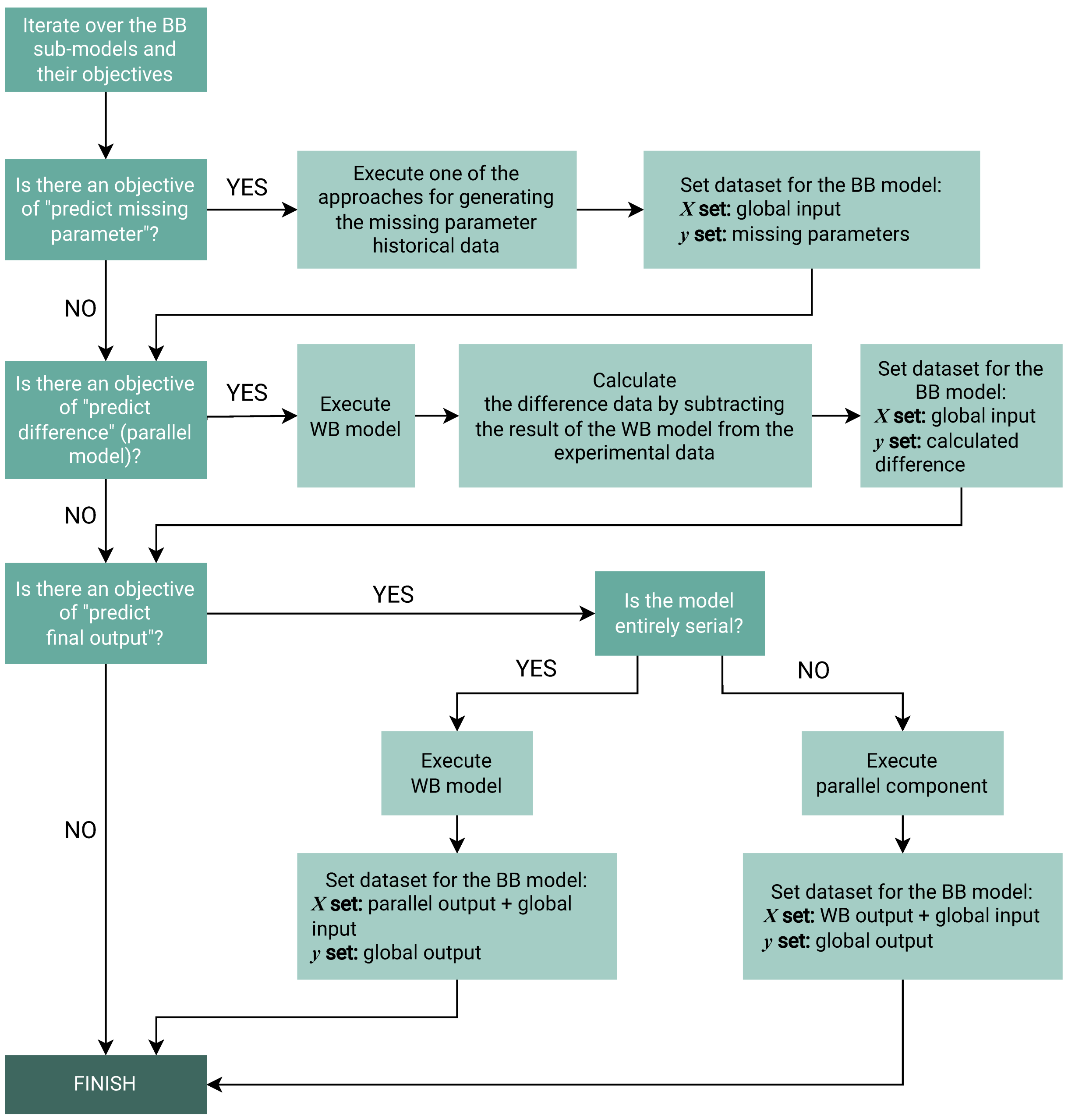
2. Create a standard ML model to identify the relation between y, X and S_d.
3. Input the training values of X and S_d to the ML model and obtain y
4. Proceed to train the official BB model with the generated y data.

Alternatively, the second approach performs the steps:

1. Iteratively try to find values for y using an optimization algorithm.
2. The idea is to find values for y that when applied to the WB model produce the closest value to the S_d training value.

Taking into account the considerations about the placement of the BB model and the execution procedure of the different WB models, a workflow that describes the training process of a hybrid model is displayed in Figure 21. The implementation of this process is reserved for a future task within the OptiMassKI project when the required production data is available, as outlined in section 6.2.

Figure 21 – Proposed training process of a hybrid model.


Iterate over the BB sub-models and their objectives
Is there an objective of "predict missing parameter"?
YES
Execute one of the approaches for generating the missing parameter historical data
Set dataset for the BB model: X set: global input y set: missing parameters
NO
Is there an objective of "predict difference" (parallel model)?
YES
Execute WB model
Calculate the difference data by subtracting the result of the WB model from the experimental data
Set dataset for the BB model: X set: global input y set: calculated difference
NO
Is there an objective of "predict final output"?
YES
Is the model entirely serial?
YES
NO
Execute WB model
Execute parallel component
Set dataset for the BB model: X set: parallel output + global input y set: global output
Set dataset for the BB model: X set: WB output + global input y set: global output
NO
FINISH

Source: Author.

5 CODE IMPLEMENTATION

After defining the concept of the configuration framework in the previous chapter, the present chapter concerns the implementation of the workflow in Python scripts. An object-oriented approach is used to represent the major elements of the framework. Thus, each created class is detailed. Finally, the created package is put to the test by executing the workflow using information from the use case.

5.1 TOOLS AND TECHNOLOGIES STACK

As stated in section 1.2, the main product of the OptiMassKI project is a software tool that handles the creation and usage of hybrid ML models. Thus, after creating the concept of the configuration framework, the next step is to develop a computer program to put it into practice.

The chosen programming language to implement the functionalities was Python. Aside from being easy to use, it is the most famous language for data science and ML. Another advantage of Python is its object-oriented structure, which will be used to develop the desired steps of the workflow, as outlined in the next section.

In future works, FastAPI and AutoML tools should also be used in the OptiMassKI project. FastAPI is a framework for building APIs with Python and will be used to make the communication between the backend and frontend interfaces of the software tool. AutoML is a concept in which the process of developing a ML model is completely automated, and it will be employed in the training process of the BB models. The most used AutoML libraries in Python are Auto-sklearn and Auto-PyTorch.

Given that these tools are known for being user-friendly and having easy implementation, their integration with the module developed in this thesis should be uncomplicated and require minimal effort.

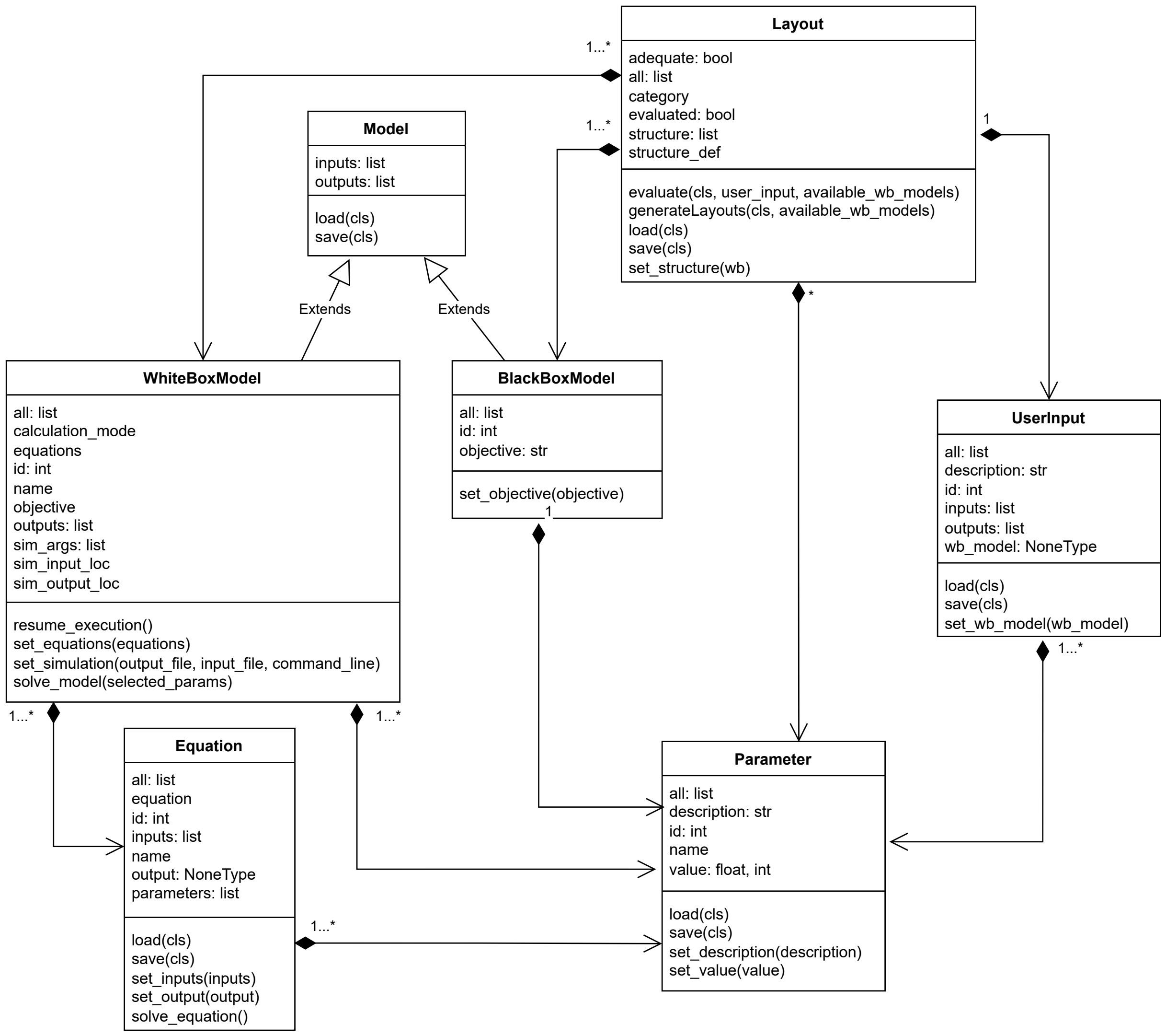
5.2 OBJECT-ORIENTED DESIGN

An object-oriented approach was employed to implement the configuration module in Python. This procedure facilitated the design and development of the workflow by organizing the system into modular and reusable components.

Object-Oriented Programming (OOP) is a concept in which the software is organized around objects, instead of functions and logic. The main component of a OOP structure is the *classes*, data types that characterize objects, including methods and attributes. The *objects* are the instances of classes, with specifically assigned data. *Methods* are functions of a class that defined certain behaviors and *attributes* are used to represent the state of the object.

The class diagram of the configuration module is depicted in Figure 22.

Figure 22 – Class diagram of the configuration module.



Source: Author.

Each created class represents an important instance within the configuration workflow, as described in the subsequent topics. They were built respecting the requirements stated in the previous chapter, in order to achieve the expected functionality.

Aside from their own functionalities, every class has the *save* and *load* methods. They are responsible for storing and retrieving the existent objects from pickle files.

A pickle file has the extension *'.pkl'* and is used to store serialized Python objects in a binary format. It retains the structure and data of the original objects and can be used with various types, including classes and instances.

The code that creates the *save* and *load* methods is very similar for each class. An example of how it is done in the Parameter class is depicted in Code Listing 1.

Code Listing 1 – Definition of the *save* and *load* methods for the Parameter class.

```python
@classmethod
def save(cls):
    file_name = "./data/" + cls.__name__ + ".pkl"

    with open(file_name, 'wb') as f:
        pickle.dump((cls.all, cls._existing_names,
    cls._id_counter), f)

@classmethod
def load(cls):
    file_name = "./data/" + cls.__name__ + ".pkl"
    try:
        with open(file_name, 'rb') as f:
            cls.all, cls._existing_names, cls._id_counter =
    pickle.load(f)
    except FileNotFoundError:
        cls._existing_names = set()
        cls._id_counter = 0
        cls.all = []
```

In the *save* method, the pickle file is opened and the class attributes, that store all the necessary information, are stored in it. Then, in the *load* method, the same file is opened and the contained information is loaded. If the file has not yet been created, the class attributes are initialized with empty values.

5.2.1 *Parameter* class

The *Parameter* class is used to represent all the variables and constants involved in the hybrid model. It is later used to compose the inputs, outputs, and equations.

Parameter has three class attributes, *_existing_names*, *_id_counter*, and *all*, that support the logic of creating new instances. To initiate a new *Parameter* object, its name must be provided. A conference is made to make sure there is no other instance registered with the same name and the object is created by setting its attributes *id*, *name*, *value*, and *description* and updating the class's attributes. Then the attributes can be set later using the methods *set_value* and *set_description*, respectively.

This logic is implemented as shown in Code Listing 2.

Code Listing 2 – Definition of the *Parameter* class.

```python
class Parameter:
    _existing_names = set()
    _id_counter = 0
    all = []

    def __init__(self, name: str) -> None:
        if name in self._existing_names:
            raise ValueError(f"A parameter with the name
    '{name}' already exists")

        self.id = self._id_counter
        self.name = name
        self.value = 0 # default value
        self.description = "" # default value

        Parameter.all.append(self)
        self._existing_names.add(name)

    def set_value(self, value: float) -> None:
        try:
            self.value = float(value)
        except ValueError:
            print(f"Error: '{value}' is not a valid number")

    def set_description(self, description: str) -> None:
        self.description = description
```

5.2.2 *Equation* class

This class is used to describe the equations that form the WB model of type "mathematical model". To create a new instance of the *Equation* class, it is necessary to provide:

- The name of the equation, as a string.
- A string containing the equation, where all the parameters are limited by brackets and the string is the result of the equation when the output variable is isolated (e.g. `"[a]*[x]+[b]"` should be the string associated to equation $y = a \cdot x + b$, given that y is the output).
- The inputs of the equation, as a list of *Parameter* instances. They can initially have the default value.
- The output of the equation, as a *Parameter*.

When an instance of Equation is initialized, the parameters of the given equation are extracted, by looking for the characters inside brackets. Then, all of the instance's attributes are defined. This happens inside the *__init__* method, which can be seen in Code Listing 3.

Another essential method shown in the code snippet is *solve_equation*. It replaces the parameters in the equation for their values and proceeds to solve it, storing the result as the equation's output. It returns an error if all the parameters have empty values, indicating that they should have attributed values in order for the equation to be solved.

Code Listing 3 – Definition of the *Equation* class.

```python
class Equation:
    _existing_names = set()
    _id_counter = 0
    all = []

    def __init__(self, equation: str, name:str, inputs:List[Parameter]=[], output:Parameter=None) -> None:

        params = []
        params_names = [match[1:-1] for match in re.findall(r"\[.*?\]", equation)]
        for p in params_names:
            params.append(p)

        self.id = self._id_counter
        self.equation = equation
        self.parameters = params
        self.name = name
        self.inputs = inputs
        self.output = output

    def solve_equation(self) -> float:
        equation = self.equation
        input_sum = 0
        for i in self.inputs:
            equation = equation.replace(f"[{i.name}]", str(i.value))
            input_sum += i.value

        if input_sum > 0:
            result = eval(equation, {"sqrt": sqrt})

            print(f"{self.output.name} = {self.equation}")
            print(f"{self.output.name} = {equation}")
            print(f"{self.output.name} = {result}")

            self.output.value = result
        else:
            raise ValueError(f"Please make sure that the inputs have been provided values.")
```

5.2.3 *Model* class

The *Model* class serves to define the sub-models within the hybrid structure. Its initialization method, as demonstrated by Code Listing 4, is uncomplicated. This class acts as a parent class for both the *BlackBoxModel* and *WhiteBoxModel* classes, which encompass more complex functionalities.

Code Listing 4 – Definition of the *Model* class.

```
class Model:

    def __init__(self, inputs:List[Equation]=[], outputs:List[Equation]=[]) -> None:
        self.inputs = inputs
        self.outputs = outputs
```

BlackBoxModel is also a straightforward class, as outlined in Code Listing 5. Its initialization method inherits from the parent class, where two lists of *Parameter* instances are assigned as the inputs and output of the sub-model, and initial values are set for the remaining attributes. Moreover, the class includes the *set_objective* method, which is used to select the objective of the BB model.

Code Listing 5 – Definition of the *BlackBoxModel* class.

```
class BlackBoxModel(Model):
    _id_counter = 0
    all = []

    def __init__(self, inputs:List[Equation]=[], outputs:List[Equation]=[]) -> None:
        super().__init__(inputs, outputs)
        self.id = self._id_counter
        BlackBoxModel.all.append(self)

    def set_objective(self, objective:str):
        self.objective = objective
```

The second subclass of *Model* is the *WhiteBoxModel* class, which is used to characterize the parametric sub-model of the hybrid structure.

As outlined in Code Listing 6, the initialization of this class is similar to the others, as it defines the main attributes instance: *inputs*, *outputs*, *calculation_model*, *id*, *name* and *objective*. Before attributing the given *calculation_model*, it is necessary to assure that it is within the valid options: "equations", "expert-knowledge", "auto-simulation" or "manual-simulation".

Code Listing 6 – Initialization of the *WhiteBoxModel* class.

```
class WhiteBoxModel(Model):
    _id_counter = 0
    all = []

    def __init__(self, calculation_mode:str, name:str ,
objective: str, inputs:List[Parameter]=[],
outputs:List[Parameter]=[]) -> None:
        super().__init__(inputs, outputs)

        calculation_types = ["equations", "expert-knowledge",
"auto-simulation", "manual-simulation"]
        assert calculation_mode in calculation_types, "The
calculation mode should be chosen between 'equations',
'expert-knowledge', 'auto-simulation' or
'manual-simulation'."

        self.id = self._id_counter
        self.calculation_mode = calculation_mode
        self.name = name
        self.objective = objective

        WhiteBoxModel.all.append(self)
```

This class includes other relevant methods. The methods *set_equations* and *set_simulation* are used to set the functionalities of the WB model, as displayed in Code Listing 7. When it's of type "mathematical model", the *calculation_mode* should be set to "equations", and the collection of equations that define the model can be informed using the method *set_equations*. Alternatively, if the *calculation_mode* is defined as "auto-simulation" or "manual-simulation", the method *set_simulation* is used to define the files where the inputs and outputs of the WB model are stored. In addition, if the *calculation_mode* is "auto-simulation", it informs the command line that triggers the execution of the simulation program.

Code Listing 7 – Definition of the *set_equations* and *set_simulation* methods of the *WhiteBoxModel* class.

```
def set_equations(self, equations: Tuple[Equation]):
    assert self.calculation_mode == "equations", "The model's
calculation mode should be 'equations'"
    self.equations = equations # In the order they are going
to be evaluated

def set_simulation(self, output_file:str, input_file:str,
command_line:str=None):
    assert self.calculation_mode in ["auto-simulation",
"manual-simulation"], "The model's calculation mode should
be 'auto-simulation' or 'manual-simulation'"
    self.sim_input_loc = input_file
    self.sim_output_loc = output_file

```

```python
    if self.calculation_mode == "auto-simulation":
        assert len(command_line) > 0, "The command line should be provided."
        self.sim_args = shlex.split(command_line)
```

The *solve_model* method serves to execute the WB model. This process is different depending on the type of the model.

As shown in Code Listing 8, when the calculation mode is "equations", it is asserted that the inputs and outputs of the provided set of equations match the ones of the WB model. Then, each equation is solved, resulting in the final output.

Code Listing 8 – Definition of the *solve_model* of the *WhiteBoxModel* method class when calculation mode is "equations".

```python
if self.calculation_mode == "equations":
    assert self.inputs == self.equations[0].inputs, "The inputs of the first equation should be the same as the White Box Model"
    assert self.outputs[0] == self.equations[-1].output, "The outputs of the last equation should be the same as the White Box Model"
    for e in self.equations:
        assert len(e.inputs) > 0 and e.output != None, "Equation has no input and/or output defined"

        e.solve_equation()

    self.outputs[0].value = self.equations[-1].output.value
```

Code Listing 9 describes the case in which the calculation mode is "expert-knowledge", where the WB model represents a manual feature selection based on physical principles. To solve this type of model, it is required that the optional argument *selected_params* of the *solve_model* method is not empty, as it provides the selected parameters by the expert. Then, these given parameters are set as the outputs of the WB model.

Code Listing 9 – Definition of the *solve_model* of the *WhiteBoxModel* method class when calculation mode is "expert-knowledge".

```python

elif self.calculation_mode == "expert-knowledge":
    assert len(selected_params) > 0, "Please provide the parameters selected by the expert"
    assert all(p in self.inputs for p in selected_params), "The selected parameters should be contained in the model's inputs"

    self.outputs = []
    for s in selected_params:
        self.outputs.append(s)
```

When the calculation mode is "auto-simulation", the inputs of the WB model are stored in the previously provided input file. Subsequently, a Python *subprocess* starts, triggering the execution of the simulation program, and the results are read from the provided output file into the output value of the WB model. Code Listing 10 displays the implementation of this logic.

Code Listing 10 – Definition of the *solve_model* of the *WhiteBoxModel* method class when calculation mode is "auto-simulation".

```python
elif self.calculation_mode == "auto-simulation":
    with open(self.sim_input_loc, mode='w', newline='') as file:
        writer = csv.writer(file)

        writer.writerow(['name', 'value'])

        for i in self.inputs:
            writer.writerow([i.name, i.value])

    subprocess.run(self.sim_args)

    with open(self.sim_output_loc, mode='r') as file:
        reader = csv.reader(file)
        next(reader)
        index = 0
        for row in reader:
            name, value = row
            assert self.outputs[index].name == name, "Output does not match expected parameter"
            self.outputs[index].value = value
            index += 1
```

The case in which the calculation mode is "manual-simulation" is similar to the previous one, with the difference that the execution of the simulation program is not automatically triggered. As represented in Code Listing 11, in the *solve_model* method, the only action is storing the inputs of the WB model in the previously provided input file. Then, the method *resume_execution*, defined in Code Listing 12, is used to read the outputs provided by the simulation and associate them to the WB model. This last method should be called after the execution of the simulation program.

Code Listing 11 – Definition of the *solve_model* method of the *WhiteBoxModel* class when calculation mode is "manual-simulation".

```python
elif self.calculation_mode == "manual-simulation":
    with open(self.sim_input_loc, mode='w', newline='') as file:
        writer = csv.writer(file)
```

```

            writer.writerow(['name', 'value'])

            for i in self.inputs:
                writer.writerow([i.name, i.value])

        print("Resume the execution after running the simulation.")
```

Code Listing 12 – Definition of the *resume_execution* method of the *WhiteBoxModel* class.

```
def resume_execution(self):
    with open(self.sim_output_loc, mode='r') as file:
        reader = csv.reader(file)
        next(reader)
        index = 0
        for row in reader:
            name, value = row
            assert self.outputs[index].name == name, "Output
    does not match expected parameter"
            self.outputs[index].value = value
            index += 1
```

5.2.4 *UserInput* class

The *UserInput* class is used to represent the use case requirements provided by the user at the beginning of the configuration workflow. Some attributes are used to characterize the user input and should be provided when creating a new instance of this class:

- The *inputs* of the hybrid model, as a list of *Parameter* objects. This is used to describe the available data.
- The *outputs* of the hybrid model, as a list of *Parameter* objects. This should represent the variable(s) that the user requires the hybrid model to predict.
- The WB model associated with the use case, as an instance of the *WhiteBox-Model* class.
- A string containing a description for the hybrid model.

These attributes can be set in the initialization of a *UserInput* instance. Moreover, the assigned WB model can also be set using the *set_wb_model* method.

The definition of this class is given by Code Listing 13.

Code Listing 13 – Definition of the *UserInput* class.

```
class UserInput:
    _id_counter = 0
    all = []

    def __init__(self, inputs:list=[], outputs:list=[],
                 description:str="",
  selected_wb_model:WhiteBoxModel=None) -> None:

        self.id = self._id_counter

        self.inputs = inputs
        self.outputs = outputs
        self.description = description
        self.selected_wb_model = selected_wb_model

    def set_wb_model(self, selected_wb_model:WhiteBoxModel) ->
  None:
        self.selected_wb_model = selected_wb_model
```

5.2.5 *Layout* class

The *Layout* class holds a big part of the functionalities of the configuration workflow. It is used to represent the generation and evaluation of the hybrid model structures, as well as the characterization of each sub-model.

There are some attributes that must be set when a *Layout* instance is created. These are:

- A string describing the *category* of the layout. It can be "serial", to represent the structures that are purely serial, or "parallel" to portray the structures that contain any parallel component.
- A string *structure_def* describing the structure of the hybrid model.
- A list of *WhiteBoxModels* and *BlackBoxModels* that compose the *structure* of the model.
- A boolean flag *evaluated*, that indicates if the layout has been already evaluated.
- A boolean flag *adequate*, that indicates a model that has already been evaluated and matches the use case requirements.

Code Listing 14 portrays the initialization of a *Layout* object, defined by the *__init__* method.

Code Listing 14 – Initialization method of the *Layout* class.

```
def __init__(self, category: str, structure_def:str,
  structure:list=[], evaluated:bool=False,
  adequate:bool=True) -> None:

        self.category = category
        self.structure_def = structure_def
        self.structure = structure
        self.evaluated = evaluated
        self.adequate = adequate

        Layout.all.append(self)
```

The functionality of generating hybrid model layouts is defined by the class method *generateLayouts*, as shown in Code Listing 15.

In this method, first, the quantity of *spaces* is defined based on the amount of available white models. For the use case of this thesis, there is always only one available WB model, but this function is flexible for other situations. The *spaces* represent the maximum quantity of sub-models that the hybrid model can have, and it is defined as twice the amount of WB models plus one. Each space will later be occupied by a BB model, a WB model, or a parallel component, or it can also be left blank.

Then, the string *models* is created to indicate all the sub-models that can be used in the hybrid model structure. The term 'X' is used to represent the part that will hold the WB model, which can be a single WB model or a parallel component, and the term 'B' represents a BB model.

Subsequently, a permutation is executed on the *model* strings, to generate all the possible combinations of the sub-models. The result is reviewed so that there is no structure with two consecutive BB models.

Thereafter, the *Layout* objects are generated. Each combination, which has an 'X' term, will originate two layouts: one of type "serial", in which the 'X' term is replaced by 'W' to represent a single WB model, and one of type "parallel", where the 'X' term is replaced by 'P' to describe a parallel component. Finally, a *Layout* containing only a parallel structure is also generated.

Code Listing 15 – Definition of the class method *generateLayouts* of the *Layout* class.

```python
@classmethod
def generateLayouts(cls, available_wb_models: list):

    n = len(available_wb_models)
    print(f'Generating layouts using {n} wb models')

    # Defining the amount of spaces
    spaces = n*2 + 1

    # Setting the models that can be combined
    models = 'X'*n + (spaces - n)*'B'

    permutations = list(set(itertools.permutations(models)))

    possible_configurations = [''.join(permutation) for permutation in permutations]

    for i in range(len(possible_configurations)):
        while 'BB' in possible_configurations[i]:
            possible_configurations[i] = possible_configurations[i].replace('BB', 'B')

    for config in possible_configurations:
        serial_structure = config.replace('X', 'W')
        parallel_serial_structure = config.replace('X', 'P')

        Layout(category='serial',
               structure_def=serial_structure)
        Layout(category='parallel',
               structure_def=parallel_serial_structure)

    # Adding a structure with only a paralel model
    Layout(category='parallel', structure_def='P')

    print('Generated layouts:')
    for l in Layout.all:
            print(f'{l.structure_def} - {l.category} ')
```

To obtain a list of *WhiteBoxModel* and *BlackBoxModel* objects based on the structure definition of each generated layout, the *set_structure* method is defined, as shown in Code Listing 16. This method creates a *WhiteBoxModel* or *BlackBoxModel* object to represent the sub-model in each space. The parallel component is represented by a list containing a WB model and a BB model.

Code Listing 16 – Definition of the *set_structure* method of the *Layout* class.

```python
def set_structure(self, wb:WhiteBoxModel):
    self.structure = []
    for model in self.structure_def:
        if model == 'P':
            self.structure.append([wb, BlackBoxModel()])
        elif model == 'W':
            self.structure.append(wb)
        elif model == 'B':
            self.structure.append(BlackBoxModel())
        else:
            pass
```

The *Layout* class has a last method, *evaluate*, that serves to analyze each structure that was previously generated, assess if it meets with the use case requisites, and characterize its sub-models. It respects the concept and steps defined in subsection 4.3.3 and subsection 4.3.4.

The first step in this method, as shown in Code Listing 17, is to check the two questions: "Are all the required parameters for the WB model available as process data?" and "Is the output of the WB model the same as the user-required output for the entire hybrid model?".

Code Listing 17 – Evaluation of the two questions that guide the evaluation of the generated hybrid models.

```
# Check if all wb parameters are contained in the set of
    parameters given as available by the user
all_params_available =
    set(wb.inputs).issubset(user_input.inputs)
# Check if the output of the wb model is the same as the one
    defined by the user
same_outputs = wb.outputs == user_input.outputs
```

Then, as portrayed in Code Listing 18, the code proceeds to create an initial version of a list of adequate layouts, based on the result of the first verification. The list *Layout._adequate_layouts* is a class attribute and starts empty. As each generated layout is evaluated, its *adequate* attribute may be altered, and the *Layout._adequate_layouts* list may be populated.

Code Listing 18 – First assessment of the generated hybrid model layouts.

```
if all_params_available:
    # Remove options with a BB model preceding a single
    # wb model or a parallel combination
    for l in Layout.all:
        if l.structure_def[0] == 'B':
            l.adequate = False
        else:
            Layout._adequate_layouts.append(l)
            l.set_structure(wb)
else:
    # Remove options without a BB model preceding a single
    # wb model or a parallel combination
    for l in Layout.all:
        if l.structure_def[0] != 'B':
            l.adequate = False
        else:
            Layout._adequate_layouts.append(l)
            l.set_structure(wb)

            assert isinstance(l.structure[0], BlackBoxModel)

            # inputs of l[0] = given parameters
            l.structure[0].inputs = user_input.inputs
            # outputs of l[0] = missing parameters
            l.structure[0].outputs = [i for i in wb.inputs if i not in user_input.inputs]

            l.structure[0].set_objective("predict missing parameter")
```

Subsequently, the generated list of adequate layouts is set to be re-evaluated under the perspective of the second question, as described by Code Listing 19. For each layout, it may be removed from the list if the WB output is not the same as the one required by the user, and there is not a BB model following a single WB model or a parallel combination. The involved BB models are also characterized by their inputs, outputs, and objectives. Finally, the code characterizes the BB model placed in parallel to the WB model in every layout of "parallel" category.

Code Listing 19 – Refinement of the list of adequate hybrid model layouts.

```
for l in Layout._adequate_layouts:
    if not same_outputs and l.structure_def[-1] != 'B':
        l.adequate = False
        Layout._adequate_layouts.remove(l)
    else:
        if l.category == 'serial':
            # B after W predicts the final result based on the output of W
            i = l.structure_def.index('W') + 1

            try:
                l.structure[i].inputs = wb.outputs + user_input.inputs
                l.structure[i].outputs = user_input.outputs
                l.structure[i].set_objective("predict final output")
            except:
                pass
        elif l.category == 'parallel':
            # Defining the inputs and outputs of the bb model
            # when in parallel with the wb model
            pi = l.structure_def.index('P')
            bb_model = l.structure[pi][1]
            bb_model.set_objective("predict difference")
            bb_model.inputs = wb.inputs
            try:
                bb_model.outputs = [Parameter('diff')]
            except:
                for p in Parameter.all:
                    if p.name == "diff":
                        bb_model.outputs = p

            # B after P predicts the final result based on the output of (W + B)
            try:
                i = l.structure_def.index('P') + 1
                print('wb', l.structure[i-1][0].outputs)
                print('bb', l.structure[i-1][1].outputs)
                l.structure[i].inputs = l.structure[i-1][0].outputs + l.structure[i-1][1].outputs
                l.structure[i].outputs = user_input.outputs
                l.structure[i].set_objective("predict final output")
            except:
                pass

        l.evaluated = True
```

5.3 VALIDATION AND TESTING

To confirm that the code implementation of the configuration module is adequate, it is necessary to test the developed classes in an analogous situation to the use case. Thus, the workflow presented in section 4.3 is going to be put into practice using the created methods.

Prior to the coding, the use case must be addressed. As discussed previously, the main objective is to predict the shape deviation in the hot forming process. In this context, the available input parameters and the desired outputs are described in Table 1.

Table 1 – Available input and desired output for the use case.

Input		**Output**	
Parameter	**Description**	**Parameter**	**Description**
$T_{Mold}(t)$, T_{Glass}	Time series data of temperature		
$F(t)$	Time series data of pressure (force)		
T_{Heat}	Heating temperature		
t_{Heat}	Heating time	S_d	Shape deviation
v	Pressing speed		
t_{hold}	Holding time		
μ, n_∞, T_g, m, E_1, T, t	Simulation parameters		

Source: Author.

In addition, the correspondent WB model should be described. It is executed manually through an external simulation, and the involved parameters are defined in Table 2.

By comparing both tables, some conclusions can be made. The first is that there are some required inputs for the WB model that are not available in the inputs provided by the user. This indicates that a BB model should be employed to predict the missing parameters, and it should precede the WB model in the hybrid structure.

Table 2 – Inputs and outputs of the WB model employed in the use case.

Input	**Output**
μ	
v	
n_∞	
T_g	
m	
E_1	S_d (Shape deviation)
T	
t	
$A(\wp)$	
$B(\wp)$	
$T_0(\wp)$	

Source: Author.

Furthermore, the output of the WB model is the same as the desired output by the user. Thus, there is no necessity for a BB model to follow the WB model in the hybrid structure, although this can be an approach to predict the noise of the final output.

With these definitions, it is possible to apply the use case to the configuration module code. The first step is to create all the necessary *Parameter* instances, as done in Code Listing 20.

Code Listing 20 – Initialization of the necessary *Parameter instances* for the use case.

```
T_mold = Parameter('T_mold')
T_glass = Parameter('T_glass')
F = Parameter('F')
T_heat = Parameter('T_heat')
t_heat = Parameter('t_heat')
v = Parameter('v')
t_hold = Parameter('t_hold')
mu = Parameter('mu')
n_infty = Parameter('n_infty')
T_g = Parameter('T_g')
m = Parameter('m')
E_1 = Parameter('E_1')
T = Parameter('T')
t = Parameter('t')
A = Parameter('A')
B = Parameter('B')
T_0 = Parameter('T_0')
S_d = Parameter('S_d')
```

Subsequently, the correspondent WB model and user input are defined, as shown in Code Listing 21.

Code Listing 21 – Creation of the *WhiteBoxModel* and *UserInput* instances for the use case validation.

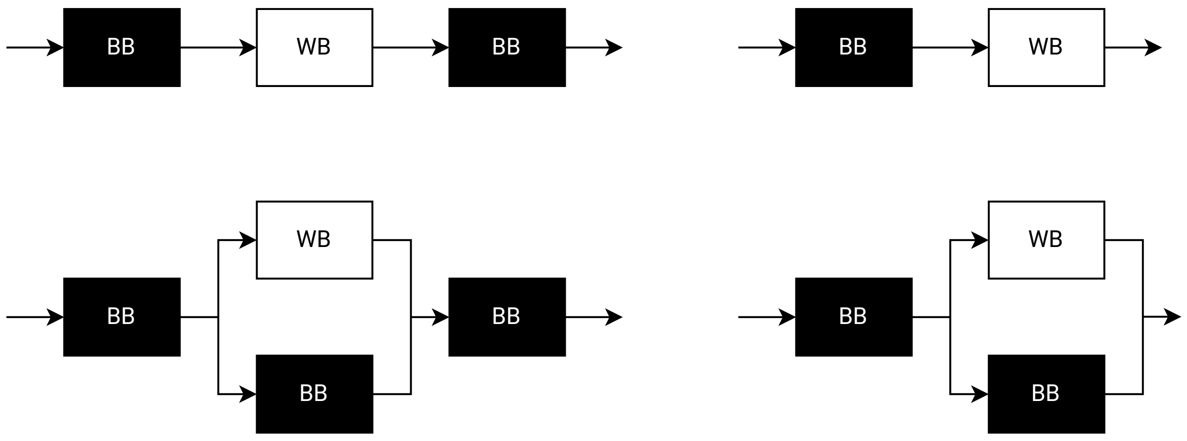
```
wb = WhiteBoxModel(calculation_mode="manual-simulation",
                   name="Material model (simulation and
                   shrinkage)",
                   goal="Predict shape deviation",
                   inputs=[mu, v, n_infty, T_g, m, E_1, T, t,
                   A, B, T_0],
                   outputs=[S_d])

user_input = UserInput(inputs=[T_mold, T_glass, F, T_heat,
                       t_heat, t_hold,mu, v, n_infty,
                       T_g, m, E_1, T, t],
                       outputs=[S_d],
                       description="Prediction of shape
                       deviation",
                       wb_model=wb)
```

Thereafter, the command `Layout.generateLayouts([wb])` is applied to generate all the possibilities of hybrid model structures, and finally, the command `Layout.evaluate(user_input=user_input, available_wb_models=[wb])` is executed to evaluate those options according to the use case description. As illustrated in Figure 23, the resultant adequate layout options are:

- The serial structure **BWB**.
- The mixed structure **BPB**.
- The serial structure **BW**.
- The mixed structure **BP**.

Figure 23 – Hybrid model structures generated by the configuration workflow in the use case validation.



Source: Author.

To analyze the characterization of the submodels, an auxiliary code was made to detail each model in the generated structures. As an example, the output of the outline of the first structure ("BWB") is shown subsequently.

```
MODEL BwB (Category: serial)

OUTLINE OF THE SUBMODELS:

Position: 0
Type: Black-box Model
Inputs: ['T_mold', 'T_glass', 'F', 'T_heat', 't_heat',
   't_hold', 'mu', 'v', 'n_infty', 'T_g', 'm', 'E_1', 'T', 't']
Outputs: ['A', 'B', 'T_0']
Objective: predict missing parameter
--------------------------------------
Position: 1
Type: White-box Model
Inputs: ['mu', 'v', 'n_infty', 'T_g', 'm', 'E_1', 'T', 't',
   'A', 'B', 'T_0']
Outputs: ['S_d']
Objective: Predict shape deviation
--------------------------------------
Position: 2
Type: Black-box Model
Inputs: ['S_d', 'T_mold', 'T_glass', 'F', 'T_heat', 't_heat',
   't_hold', 'mu', 'v', 'n_infty', 'T_g', 'm', 'E_1', 'T', 't']
Outputs: ['S_d']
Objective: predict final output
```

6 CONCLUSION

The final chapter of this document concludes the presented work and outlines the next tasks within the OptiMassKI project. The first section provides a conclusive summary of the bachelor thesis project, highlighting the accomplishment of the proposed objectives. Then, in the second section, the future works are described, as they provide continuity to the developed configuration framework.

6.1 CONCLUSIVE SUMMARY

This document presented the development of a configuration framework that creates hybrid models as an approach to optimize the hot forming process in the production of thin glass. This is an important step within the OptiMassKI project, which aims to create a software tool that supports SMEs to improve their production process using data-driven solutions, without the requirement for a specialized data science team.

As a result of a literature survey, it was concluded that there are no known applications of hybrid models to the required use case, although this is an approach that has existed for many years and has been successfully applied to other situations. Nonetheless, the research was valuable to provide information on the theory behind hybrid models and how to define their structure.

Based on the gathered information and on the requirements of the use case, the concept of the configuration framework was created. A comprehensive workflow was developed, describing generally how the process should occur and detailing the creation of a White Box Library, and the generation and evaluation of hybrid model structures. Furthermore, considering the estimated limitations concerning the availability of data, a guideline was provided on how to train the hybrid models, which is the following step to the work presented in this document.

Thereafter, a Python package was created to implement the concept of the configuration framework. An object-oriented approach was used to describe the important instances of the process. Finally, this development was validated by going through the workflow steps using the information on the use case.

Taking these outcomes into consideration, it is possible to conclude that the main objectives of the bachelor thesis project were achieved. The developed framework is capable of generating hybrid model structures automatically and can be applied to different situations, without the need for the user to be a data science specialist. This work is one of the central steps of the OptiMassKI project and has provided a valuable result to base future developments.

6.2 FUTURE WORK

There are some tasks within the OptiMassKI project that can already be outlined and will provide continuity to the project presented by this document. These tasks concern the training and evaluation of the generated hybrid model structures and the creation of the user interface that will support the functionalities.

6.2.1 Training and evaluating the hybrid models using AutoML

The immediate next step to the work developed in this thesis involves applying an AutoML tool to train the BB parts of the hybrid model. This training process should be integrated into the existing code to ensure it aligns with the entire Hybrid Model Configuration process.

In addition, it is necessary to define evaluation metrics that take variables like accuracy, error, processing time, and computational effort into account, enabling the selection of the best-performing hybrid model structure. Subsequently, a code covering the usage phase of the tool, as described in Figure 1, also needs to be developed.

To train the hybrid models and validate both the configuration and usage processes, historical process data is required to execute the model and ensure its effectiveness.

6.2.2 Creation of user interface to configure and use the hybrid model

Another important future task of the OptiMassKI project encompasses the development of a front-end software to sustain the constructed functionalities. This user interface should cover both the configuration and usage phases.

In the configuration phase, the tool should enable tasks such as feeding the WB Library with new models, setting up use case requirements (including general inputs, outputs, available data, and selected WB models), and providing notifications to the user when simulations are running and the process needs to be restarted. The interface should also allow for visualizing the progress of hybrid model configuration and evaluating trained hybrid models, with the ability to manually change the chosen model.

In the usage phase, the interface should enable users to input process data and easily visualize the obtained results.

BIBLIOGRAPHY

AGUIAR, Helena Cristina; FILHO, Rubens Maciel. Neural network and hybrid model: a discussion about different modeling techniques to predict pulping degree with industrial data. **Chemical Engineering Science**, Elsevier BV, v. 56, n. 2, p. 565–570, Jan. 2001. DOI: 10.1016/s0009-2509(00)00261-x. Available from: https://doi.org/10.1016/s0009-2509(00)00261-x.

BBC, Research. Light Detection and Ranging (LiDAR): Technologies and Global Markets. **Research and Markets**, 2021. Available from: https://www.researchandmarkets.com/reports/5300863/light-detection-and-ranging-lidar.

BLIEDTNER, J.; GRÄFE, G. **Optiktechnologie: Grundlagen - Verfahren - Anwendungen - Beispiele ;** [s.l.]: Fachbuchverl. Leipzig im Carl-Hanser-Verlag, 2008. ISBN 9783446408968.

BRAUER, Delia S.; RÜSSEL, Christian; KRAFT, Jörg. Solubility of glasses in the system P2O5–CaO–MgO–Na2O–TiO2: Experimental and modeling using artificial neural networks. **Journal of Non-Crystalline Solids**, v. 353, n. 3, p. 263–270, 2007. ISSN 0022-3093. DOI: https://doi.org/10.1016/j.jnoncrysol.2006.12.005.

CAI, Shengze; MAO, Zhiping; WANG, Zhicheng; YIN, Minglang; KARNIADAKIS, George Em. Physics-informed neural networks (PINNs) for fluid mechanics: a review. **Acta Mechanica Sinica**, Springer Science and Business Media LLC, v. 37, n. 12, p. 1727–1738, Dec. 2021. DOI: 10.1007/s10409-021-01148-1. Available from: https://doi.org/10.1007/s10409-021-01148-1.

DORISSEN, Jonas; HEYMANN, Henrik; SCHMITT, Robert H. Hybrid ML for Parameter Prediction in Production. **16th CIRP Conference on Intelligence Computation in Manufacturing Engineering**, 2022.

FELDMANN, Markus; KASPER, Ruth; LANGOSCH, Katharina. **Glas für tragende Bauteile**. [S.l.: s.n.], Jan. 2012. ISBN 978-3-8041-1626-9.

GÉRON, Aurélien. **Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools and techniques to build Intelligent Systems**. [S.l.]: O'Reilly Media, Inc, 2019.

GLASSEY, Jarka; STOSCH, Moritz von (Eds.). **Hybrid Modeling in Process Industries**. [S.l.]: CRC Press, Feb. 2018. DOI: 10.1201/9781351184373. Available from: https://doi.org/10.1201/9781351184373.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 349, n. 6245, p. 255–260, 2015. DOI: 10.1126/science.aaa8415.

KERSTING, Kristian. Machine Learning and Artificial Intelligence: Two fellow travelers on the quest for intelligent behavior in machines. **Frontiers in Big Data**, v. 1, 2018. DOI: 10.3389/fdata.2018.00006.

KRAUS, M. A.; DRASS, M. Artificial Intelligence for Structural Glass Engineering Applications — Overview, case studies and future potentials. **Glass Structures & Engineering**, v. 5, n. 3, p. 247–285, 2020. DOI: 10.1007/s40940-020-00132-8.

LIANG, Qiaokang; XIANG, Shao; LONG, Jianyong; SUN, Wei; WANG, Yaonan; ZHANG, Dan. Real-time comprehensive glass container inspection system based on deep learning framework. **Electronics Letters**, v. 55, n. 3, p. 131–132, 2019. DOI: https://doi.org/10.1049/el.2018.6934.

LIU, Han; FU, Zipeng; YANG, Kai; XU, Xinyi; BAUCHY, Mathieu. Machine learning for glass science and engineering: A review. **Journal of Non-Crystalline Solids**, v. 557, p. 119419, 2021. ISSN 0022-3093. DOI: https://doi.org/10.1016/j.jnoncrysol.2019.04.039.

MCCARTHY, John. **What is Artificial Intelligence?** Stanford, CA, 2007. Available from: http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html.

MENDE, Hendrik; FRYE, Maik; VOGEL, Paul-Alexander; KIRORIWAL, Saksham; SCHMITT, Robert H.; BERGS, Thomas. On the importance of domain expertise in feature engineering for predictive product quality in production. **16th CIRP Conference on Intelligence Computation in Manufacturing Engineering**, 2022.

MITCHELL, T.M. **Machine Learning**. [S.l.]: McGraw-Hill, 1997. (McGraw-Hill International Editions). ISBN 9780071154673.

PARK, Jisu; RIAZ, Hamza; KIM, Hyunchul; KIM, Jungsuk. Advanced cover glass defect detection and classification based on multi-DNN model. **Manufacturing Letters**, v. 23, p. 53–61, 2020. ISSN 2213-8463. DOI: `https://doi.org/10.1016/j.mfglet.2019.12.006`.

PERES, Ricardo Silva; JIA, Xiaodong; LEE, Jay; SUN, Keyi; COLOMBO, Armando Walter; BARATA, Jose. Industrial Artificial Intelligence in Industry 4.0 - Systematic Review, Challenges and Outlook. **IEEE Access**, v. 8, p. 220121–220139, 2020. DOI: `10.1109/ACCESS.2020.3042874`.

PSICHOGIOS, Dimitris C.; UNGAR, Lyle H. A hybrid neural network-first principles approach to process modeling. **AIChE Journal**, Wiley, v. 38, n. 10, p. 1499–1511, Oct. 1992. DOI: `10.1002/aic.690381003`. Available from: `https://doi.org/10.1002/aic.690381003`.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. **Journal of Computational Physics**, Elsevier BV, v. 378, p. 686–707, Feb. 2019. DOI: `10.1016/j.jcp.2018.10.045`. Available from: `https://doi.org/10.1016/j.jcp.2018.10.045`.

RAVINDER; VENUGOPAL, Vineeth; BISHNOI, Suresh; SINGH, Sourabh; ZAKI, Mohd; GROVER, Hargun Singh; BAUCHY, Mathieu; AGARWAL, Manish; KRISHNAN, N. M. Anoop. Artificial intelligence and machine learning in glass science and technology: 21 challenges for the 21st century. **International Journal of Applied Glass Science**, v. 12, n. 3, p. 277–292, 2021. DOI: `https://doi.org/10.1111/ijag.15881`.

RUDIN, Cynthia. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. **Nature Machine Intelligence**, v. 1, n. 5, p. 206–215, 2019. DOI: `10.1038/s42256-019-0048-x`.

TANDIA, Adama; ONBASLI, Mehmet C.; MAURO, John C. Machine Learning for Glass Modeling. In: **Springer Handbook of Glass**. Ed. by J. David Musgraves, Juejun Hu and Laurent Calvez. Cham: Springer International Publishing, 2019. P. 1157–1192. DOI: `10.1007/978-3-319-93728-1_33`.

THOMPSON, Michael L.; KRAMER, Mark A. Modeling chemical processes using prior knowledge and neural networks. **AIChE Journal**, Wiley, v. 40, n. 8, p. 1328–1340, Aug. 1994. DOI: `10.1002/aic.690400806`. Available from: `https://doi.org/10.1002/aic.690400806`.

VU, Anh Tuan; GULATI, Shrey; VOGEL, Paul-Alexander; GRUNWALD, Tim; BERGS, Thomas. Physics-Informed Data-Driven Models for Predicting Time- and Temperature-Dependent Viscoelastic Material Behaviors of Optical Glasses. **SSRN Electronic Journal**, Elsevier BV, 2021. DOI: `10.2139/ssrn.3822865`. Available from: `https://doi.org/10.2139/ssrn.3822865`.

VU, Anh Tuan; VOGEL, Paul Alexander; SIVA SUBRAMANIAN, Abimathi; GRUNWALD, Tim; BERGS, Thomas. Real-Time Quality Control in Thin Glass Forming Using Infrared Thermography and Deep Learning. **Key Engineering Materials**, v. 926, p. 2312–2321, Aug. 2022. DOI: `10.4028/p-5w9vr9`.

VU, Anh-Tuan; KREILKAMP, Holger; DAMBON, Olaf; KLOCKE, Fritz. Nonisothermal glass molding for the cost-efficient production of precision freeform optics. **Optical Engineering**, SPIE-Intl Soc Optical Eng, v. 55, n. 7, p. 071207, May 2016. DOI: `10.1117/1.oe.55.7.071207`. Available from: `https://doi.org/10.1117/1.oe.55.7.071207`.

WOLPERT, David H. The Lack of A Priori Distinctions Between Learning Algorithms. **Neural Computation**, MIT Press - Journals, v. 8, n. 7, p. 1341–1390, Oct. 1996. DOI: `10.1162/neco.1996.8.7.1341`. Available from: `https://doi.org/10.1162/neco.1996.8.7.1341`.

WU, Gang-Zhou; FANG, Yin; KUDRYASHOV, Nikolay A.; WANG, Yue-Yue; DAI, Chao-Qing. Prediction of optical solitons using an improved physics-informed neural network method with the conservation law constraint. **Chaos, Solitons & Fractals**, Elsevier BV, v. 159, p. 112143, June 2022. DOI: `10.1016/j.chaos.2022.112143`. Available from: `https://doi.org/10.1016/j.chaos.2022.112143`.

YAN, Jiwang; ZHOU, Tianfeng; MASUDA, Jun; KURIYAGAWA, Tsunemoto. Modeling high-temperature glass molding process by coupling heat transfer and viscous deformation analysis. **Precision Engineering**, v. 33, n. 2, p. 150–159, 2009. ISSN 0141-6359. DOI: `https://doi.org/10.1016/j.precisioneng.2008.05.005`.

ZENDEHBOUDI, Sohrab; REZAEI, Nima; LOHI, Ali. Applications of hybrid models in chemical, petroleum, and energy systems: A systematic review. **Applied Energy**, Elsevier BV, v. 228, p. 2539–2566, Oct. 2018. DOI: `10.1016/j.apenergy.2018.06.051`. Available from: `https://doi.org/10.1016/j.apenergy.2018.06.051`.

ZHOU, Tianfeng; YAN, Jiwang; YOSHIHARA, Nobuhito; KURIYAGAWA, Tsunemoto. Study on Nonisothermal Glass Molding Press for Aspherical Lens. **Journal of Advanced Mechanical Design, Systems, and Manufacturing**, Japan Society of Mechanical Engineers, v. 4, n. 5, p. 806–815, 2010. DOI: `10.1299/jamdsm.4.806`. Available from: `https://doi.org/10.1299/jamdsm.4.806`.