DEPARTMENT OF
COMPUTER SCIENCE

JOSÉ BONANÇA PEDREIRA
BSc in COMPUTER SCIENCE AND ENGINEERING

HYBPY: A WEB-BASED PLATFORM FOR HYBRID MODELING OF BIOPROCESSES

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

HYBPY: A WEB-BASED PLATFORM FOR HYBRID MODELING OF BIOPROCESSES

JOSÉ BONANÇA PEDREIRA

BSc in COMPUTER SCIENCE AND ENGINEERING

Adviser: Rafael Costa
Assistant Professor, NOVA School of Science and Technology, NOVA University Lisbon

Co-advisers: Pedro Barahona
Full Professor, NOVA School of Science and Technology, NOVA University Lisbon

Rui Oliveira
Associate Professor, NOVA School of Science and Technology, NOVA University Lisbon

Examination Committee

Chair: Name of the committee chairperson
Full Professor, FCT-NOVA

Rapporteur: Name of a rapporteur
Associate Professor, Another University

Members: Another member of the committee
Full Professor, Another University

Yet another member of the committee
Assistant Professor, Another University

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2024

HYBpy: a web-based platform for hybrid modeling of bioprocesses

Copyright © José Bonança Pedreira, NOVA School of Science and Technology, NOVA University Lisbon.
The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.18) [1].

Acknowledgements

Firstly, I want to give my gratitude to my supervisors, Rafael Costa and Pedro Barahona, for the help and support that they gave me throughout my work. Their help was crucial, especially in the writing that I am not the greatest at. I also want to give a special thanks to José Pinto for all the help and advice throughout the tool development process.

I am deeply grateful to FCT-UNL and all the professors for their unwavering commitment to excellence and outstanding mentorship, which has greatly influenced my academic journey.

I want to express my heartfelt thanks to my family, especially my mother, for being the best role model I could ask for. Her unwavering belief in me and her sacrifices have always motivated me to work hard and achieve my goals. I am also deeply grateful to my father for his support and hard work, which allowed me to pursue my studies. I thank my siblings, especially my sister, for your constant support and motivation.

To my Love, Teresa Soares, I can't thank you enough for all your help, emotional support, and patience when I needed it most. You are always there at my side. Thank you for always putting a smile on my face when the times get tough, and thanks for filling my heart with love. Without you, this journey would have been unbearable.

And finally, to my friends João Emauz, Mateus Branco, Luís Rodrigues, Catarina Matos, Catarina Filipe, António Tabarra, Hugo Lóio, Miguel Santos and Pedro Cunha for all the patience they showed during my endless ramblings about this thesis and for always being there to provide a much-needed distraction.

"If at first you don't succeed, blast them with your blue eyes again!"

— **Seto Kaiba**, Yu-Gi-Oh! Ep.173
(Duelist, genius and trillionaire)

Abstract

Hybrid models, which combine machine learning (ML) with traditional mechanistic models, are pivotal in developing digital twins for Biopharma 4.0 (i.e. the digital transformation of biopharmaceutical manufacturing, leveraging data and modern technologies to optimize processes). Digital twins—virtual replicas of physical systems—enhance process comprehension and control. In this context, hybrid modeling techniques are especially useful as they integrate prior knowledge with data availability to generate more accurate and adaptive models. They have been widely applied in process systems engineering for predictive modeling, process monitoring, model predictive control and design of experiments as well as systems biology problems. Moreover, in the last years a substantial body of work in (bio)systems approaches that merge ML/AI and traditional engineering methods were developed, namely the concept of "general bioprocess hybrid model" to describe bioprocess dynamics and for the field of systems biology.

Despite its relevance, the use of such hybrid modeling techniques has usually been limited. Only a small number of experts (research groups of the in-house tools) have the knowledge and methods to develop such models worldwide. Additionally, a specific open-source and user-friendly software tool to develop hybrid models is lacking.

This work aims to address current challenges by developing an open-source, modular, and user-friendly automated framework for building, simulating, and analysing hybrid models. The result is HYBpy, a Python tool designed to accelerate the development and training of decision hybrid models using a generalized, step-by-step pipeline. HYBpy distinguishes itself by providing a web-based interface, making it accessible to people with no programming knowledge or experience with hybrid models development. By providing this tool, we aim to add significant value to the hybrid modeling community and facilitate wider engagement and ease of use among researchers.

Keywords: hybrid modeling, Python, web-based framework, bioprocess digitalization, hybrid mechanistic/ML modeling, Industry 4.0

Resumo

Modelos híbridos, que combinam aprendizagem automática (ML) com modelos mecanísticos tradicionais, são fundamentais no desenvolvimento de gémeos digitais para a Biopharma 4.0 (ou seja a transformação digital da fabricação biofarmacêutica, aproveitando dados e tecnologias modernas para otimizar processos). Os gémeos digitais—réplicas virtuais de sistemas físicos—melhoram a compreensão e o controlo dos processos. Neste contexto, as técnicas de modelação híbrida são especialmente úteis, por integrarem conhecimento prévio com disponibilidade de dados para gerar modelos mais precisos e adaptativos. Têm sido amplamente aplicadas na engenharia de sistemas de processos para modelação preditiva, monitorização de processos, controlo preditivo baseado em modelos e desenho de experiências, bem como em problemas de biologia de sistemas. Além disso, nos últimos anos, desenvolveu-se um corpo substancial de trabalho em abordagens de (bio)sistemas que combinam ML/IA e métodos tradicionais de engenharia, nomeadamente o conceito de "modelo híbrido geral de bioprocessos" para descrever a dinâmica de bioprocessos e para o campo da biologia de sistemas.

Apesar da sua importância, o uso de tais técnicas de modelação híbrida tem sido limitado. Apenas um pequeno número de especialistas possui o conhecimento e os métodos para desenvolver tais modelos mundialmente. Além disso, existe uma procura por um 'software' "open-source" e de fácil utilização.

Este trabalho visa abordar os desafios atuais através do desenvolvimento de um 'software', para construir, simular e analisar modelos híbridos. O resultado é o HYBpy, uma ferramenta em Python projetada para acelerar o desenvolvimento e o treino de modelos híbridos, utilizando uma pipeline generalizada. O HYBpy distingue-se por fornecer uma interface web, tornando-o acessível a pessoas sem conhecimentos de programação ou experiência no desenvolvimento de modelos híbridos. Ao disponibilizar esta ferramenta, aspiramos acrescentar valor significativo à comunidade de modelação híbrida e facilitar um envolvimento mais amplo e facilidade de uso entre os investigadores.

Palavras-chave: modelação híbrida, ferramenta Python, software, digitalização de bioprocessos, modelação híbrida mecanística/ML, indústria 4.0

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

The biopharmaceutical sector, renowned for its pivotal role in healthcare innovation, is undergoing a phase of exponential growth. In 2021, the global market for biopharmaceuticals was valued at a staggering USD 389.6 billion [2] and is expected to grow at a compound annual growth rate (CAGR) of 7.6%, reaching around USD 788.4 billion by 2031 [3]. This surge is spearheaded by monoclonal antibodies, which not only constituted early sales of USD 103.4 billion in 2018 (or 55% of all biopharmaceuticals' total revenue)[4] but are also expected to surpass USD 140 billion by 2024 [5]. Due to its effectiveness in treating serious illnesses like cancer, autoimmune disorders, and infectious diseases, this class of biotherapeutics has been in high demand [6].

However, despite this rapid expansion and the ensuing demand [4, 7], the development of such biologics is typically a slow and resource-intensive process, taking up to 10–12 years and an investment of at least USD 2 billion [8, 9]. The critical tasks in biopharmaceutical process development and Chemistry Manufacturing and Control (CMC) are 1) design and optimization, 2) monitoring and control, and 3) scale-up, as shown in Figure 1.1. In a highly multidimensional design space, trial and error-based or restricted operations enforce sub-optimality and risk of not meeting quality specifications. This approach also increases the time and cost of development, mainly if it stems from inefficient transfer of past learning. Currently, knowledge transfers are performed as heuristics or conclusions determined based on observations drawn from particular cases [10].

Here lies the potential of Biopharma 4.0 to transform the landscape. Biopharma 4.0 encapsulates the integration of advanced digital technologies, data analytics, dynamic models, and automation into biopharmaceutical manufacturing and development processes. This integration aims to streamline operations, enhance efficiency, and reduce the time and cost associated with bringing new treatments to market. By leveraging these technologies, the biopharmaceutical sector can overcome existing challenges, fostering a more innovative, agile, and cost-effective approach to drug development [11].

Bioreactors, vessels where biological reactions occur, are complex multiscale processes

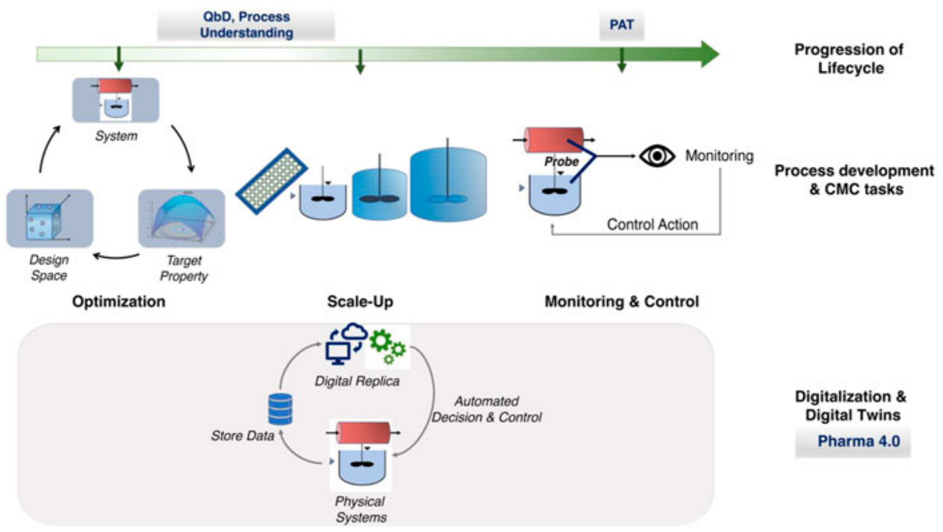


Figure 1.1: Schematic representation of the key process development and CMC tasks throughout the biopharmaceutical lifecycle. Adapted from [2].

that are very challenging to model. One of the challenges in employing model-based methods is ensuring model accuracy, precision, and predictive capabilities. There are three main types of bioreactor models: mechanistic, empirical, and hybrid models. Each will be discussed in more detail below.

The mechanistic models use mathematical equations to describe the physical, chemical, and biological processes within a bioreactor. These models are based on fundamental principles such as mass and energy conservation, reaction kinetics, and fluid dynamics. The goal of mechanistic modeling is to accurately predict the behavior of the bioreactor under various operating conditions, which can aid in process optimization, scale-up, and control [12].

The empirical models use experimental data to develop mathematical models that describe the behavior and performance of the bioreactor system [12]. These models are typically based on observed relationships between variables rather than derived from fundamental principles or theories. Machine Learning (ML) algorithms are being increasingly used to develop empirical models, as they possess a more prominent ability to capture complex, non-linear relationships in the data. ML techniques like neural networks are particularly well-suited for this, as they are more effective at capturing these complex relationships and making predictions under a wide range of conditions.

Empirical models, including those built with ML algorithms, can be used to predict the behavior of a bioreactor under different conditions, optimize its performance, or design control strategies. They are instrumental when the underlying mechanisms are complex or not fully understood, as is often the case in biological systems. They have some limitations because they are based on observed data and, therefore, may not accurately predict the system's behavior under conditions significantly different from those under which the data were collected. The empirical model's structure, developed through traditional statistical methods or modern ML approaches, is chosen as a trade-off between the amount of noise and the information the data may give [12].

The hybrid models combine different modeling approaches to understand better and predict the behavior of biological systems and processes. This approach often combines mechanistic and empirical models to leverage their strengths. In a hybrid model, mechanistic models are frequently used to describe the well-understood parts of the system. In contrast, empirical models are used where the system behavior is complex or not well-understood [13]. For example, a hybrid bioreactor model might use a mechanistic model to describe the growth of microorganisms and an empirical model to describe the effect of operational parameters on the yield of the process. Hybrid modeling is suitable for all the process goals of the biopharmaceutical industry, such as QbD and PAT [14, 15], process optimization [16, 17], scale-up [18], for process monitoring and control [16, 19], digitalization [20, 21], and digital twins [22].

1.2 Motivation

Hybrid models combine state-of-the-art machine learning algorithms (derived from data) with mechanistic models (derived from knowledge) in the same model structure. Based on recent advancements, hybrid models effectively combine the predictive power of data-driven machine learning models with the detailed insight from mechanistic models based on scientific knowledge, offering a comprehensive approach to modeling complex bioprocesses [23][24].

A study by Sharma and Liu (2021) [25] provides a comprehensive overview of hybrid process modeling and optimization, combining scientific knowledge and data analytics in bioprocessing and chemical engineering with a science-guided machine learning (SGML) approach. They classify the approach into two broad categories. The first refers to the case in which a data-based ML model complements and improves the prediction accuracy of a first-principles science-based model. In contrast, the second corresponds to the case in which scientific knowledge aids in the ML model's scientific consistency. They comprehensively review the scientific and engineering literature on the hybrid SGML approach and propose a systematic classification of hybrid SGML models. This approach underscores the potential of hybrid models in predictive modeling, process monitoring, and design of experiments within bioprocessing and chemical engineering domains.

The Corrective Source Term Approach (CoSTA), introduced by Blakseth et al. (2022) [26], further elaborates on combining physics-based models with data-driven components to correct errors in predictive models. This approach significantly advances hybrid modeling by integrating deep neural networks with traditional physics-based models, ensuring more accurate and interpretable outcomes.

Given these advancements, it's evident that hybrid modeling represents a critical frontier in bioprocessing and systems biology. The proposed Python tool aims to democratize access to this sophisticated modeling approach, offering a user-friendly web interface for constructing, analyzing, and simulating hybrid models. By leveraging Python's extensive libraries, such as NumPy, pandas, SciPy, TensorFlow, and PyTorch, this tool will facilitate the integration of mechanistic and machine learning models, making advanced hybrid modeling accessible to a broader community of researchers and practitioners interested in optimizing bioprocesses and advancing systems' biology research.

1.3 Objectives

The core ambition of this dissertation is to develop HYBpy, a Python-based tool to build and analyze hybrid models within the bioprocess domain. HYBPY is designed to operate within the Systems Biology Markup Language (SBML) framework [27], ensuring compatibility with existing biological models while facilitating their transition into hybrid models. As such, the aims of this work are the following:

- Create a Python tool capable of building, training, and simulating hybrid models for bioprocesses and complex biological systems, ensuring compatibility with SBML standards.
- Build a web-based interface to facilitate the hybrid modeling process, making it accessible to users without extensive programming experience.
- The development and employment of new training methods;
- Develop visualization interfaces to help users interpret model structures, training progress, and simulation results effectively.
- Release HYBpy as an open-source project to encourage collaboration, adoption, and contributions from the hybrid modeling community.

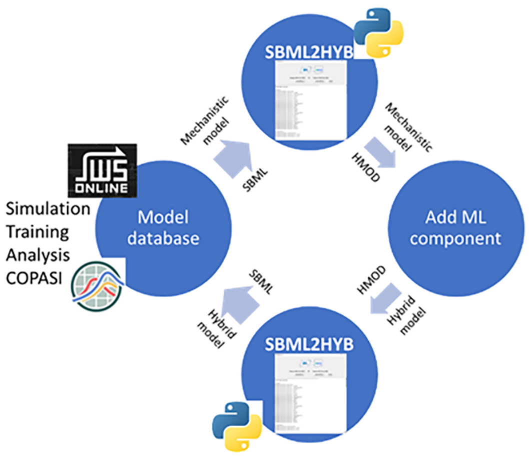


Figure 1.2: Overview of the SBML2HYB pipeline. The SBML (mechanistic) models are stored in databases like BioModels [28] and can be converted to the `.hmod` format (and vice versa) by the SBML2HYB tool [29].

1.4 Contributions

This thesis aims to significantly enhance the functionality of an existing MATLAB-based hybrid modeling code developed by members of our research group. To overcome the current limitations of the previous tool, including the dependency on proprietary software/license, this work intends to provide an open-source, modular, and easy-to-use (automated) web-based framework for hybrid model building, simulating, and analysis. The transition is not merely a shift in programming languages. Still, it represents a comprehensive upgrade aimed at making the tool more accessible, versatile, and conducive to bioprocessing while making hybrid model development more efficient and effective. The core contributions of this thesis are outlined as follows:

- Development of an open-source and user-friendly web-based framework as a bioprocess development solution: recognizing the limitations of desktop-only applications, this work introduces web-based interfaces for the hybrid modeling tool. This development aims to improve accessibility and user engagement by allowing process researchers and practitioners to interact with the tool through a web browser, facilitating a broader adoption and collaborative use.
- Enhanced file handling and visualization: a critical enhancement in the new version is the capability for users to upload and directly preview their files within the web interface. This feature significantly improves the user experience by providing immediate feedback on the data being analyzed and streamlining the model configuration and simulation process.
- Expansion of training library: a key advancement is the introduction of new training methods of hybrid models (Trust Region Reflective [30], Limited-memory BFGS [31], Trust Region Constrained [32], Dual Annealing [33]) that were not available in the original MATLAB version.

1.5 Thesis Outline

The thesis is structured to present the development process systematically, the novel features of the Python-implemented hybrid modeling tool, and the implications of these enhancements for research and development. The thesis is structured as follows.

- Chapter 1 - Introduction: deals with the contextualization of the problem by outlining the motivation behind the work, the objectives of the thesis, and the expected impact on the hybrid modeling community.
- Chapter 2 - Background: provides an overview of hybrid modeling, the current state of computational tools in biopharmaceutical research, and identifies challenges and opportunities that the upgraded tool aims to address.

- Chapter 3 - Related Work: reviews existing hybrid modeling tools, mainly the tool developed in MATLAB, to assess their utility and limitations. This analysis introduces our Python tool as a strategic enhancement in hybrid modeling techniques.
- Chapter 4 - Methodology: details the technical approaches employed in developing the Python tool and the web-based interface, including the rationale behind essential design and implementation decisions.
- Chapter 5 - Results and Discussion: evaluates the tool's performance and usability, showcasing and demonstrating its benefits with applications to three case studies. It explores the tool's impact on hybrid modeling research and addresses limitations.
- Chapter 6 - Conclusion and Future Work: summarizes the thesis and exposes how the obtained results indicate its further use as a hybrid modeling tool in bioprocesses and biological systems to build knowledge without wet-lab work.

2 Background

2.1 Hybrid Models

Hybrid models originate from the intersection of mechanistic understanding and machine learning (ML) innovation, providing a ground-breaking method for scientific research. These models are critical in domains that deal with complicated systems and processes where typical analytical methods fall short. Hybrid models achieve a previously impossible mix of precision and adaptability by combining the deterministic nature of mechanistic models—rooted in proven scientific principles—with the pattern-recognition skill of machine learning [25, 34].

The dual nature of hybrid models is essential to their appeal: they improve scientific models' predictive power and accuracy with data-derived insights while instilling scientific theories in machine learning algorithms. This synthesis provides a deeper and more precise understanding of complicated processes and assures that predictions are consistent and reliable [35]. Hybrid models provide a nuanced framework for investigating intricate dynamics and interactions in chemical engineering and environmental science, where empirical data and theoretical models are important [36].

This combination produces more precise and efficient models in computational resource utilization [37]. The motivation to merge ML with traditional scientific knowledge stems from the limitations of solely mechanistic or ML-based methodologies used alone. Hybrid models aim to overcome these hurdles by providing unique approaches for improving model performance, achieving higher resolution outputs, easing the modeling of complicated interdependencies, and even disclosing new fundamental equations [38].

The development of hybrid models represents a methodological advancement, pointing to a more integrated and flexible approach to research. These models exemplify the interdisciplinary nature of modern scientific research, bridging the gap between different academic departments to foster a thorough understanding of complex systems. Initiatives like science-guided machine learning (SGML) [25] models show this integrated approach by utilizing hybrid modeling methodologies across various domains such as bioprocess and chemical engineering. These efforts emphasize hybrid models' ability to improve prediction accuracy, maintain scientific integrity, and widen the breadth of ML applications in research pursuits.

The significance of hybrid models extends beyond their immediate scientific contributions. By facilitating an interface between empirical data analysis and theoretical model construction, they embody a paradigm shift in how we approach problem-solving and knowledge discovery. This innovative melding of methodologies propels scientific and engineering fields forward and offers a template for addressing the multifaceted challenges of modern scientific and technological landscapes [25].

2.2 Historical Development of Hybrid Models

Hybrid modeling dates back to the early 1990s, representing a fundamental paradigm shift in computational and process engineering methodologies. Psichogios and Ungar's seminal work in 1992 established the notion of hybrid modeling, setting the platform for future breakthroughs in this subject. They separated hybrid models into two architectures: serial and parallel [16, 23]. This categorization has been pivotal in guiding subsequent research and application of hybrid models across various scientific disciplines.

In the serial architecture, fragments of knowledge-based models are augmented with data-driven models, allowing for the integration of empirical data analysis into traditionally deterministic models. This approach facilitates a more refined understanding of the system under study by leveraging the strengths of both model types. On the other hand, the parallel architecture involves the simultaneous setup of knowledge-based and data-driven models, with an aggregation of predictions from both considered for final analysis [24]. This architecture aims to harness the complementary capabilities of both models, providing a comprehensive view of the system's dynamics.

Since its inception, numerous studies have demonstrated the implementation of hybrid modeling for various bio-processes, predominantly utilizing the serial architecture [15]. This preference is attributed to the serial architecture's ability to constrain and segregate the information learned by the data-based model. This strategy ensures that the empirical learning process is directed and informed by established knowledge, reducing the hazards associated with data-driven models utilized in isolation. In contrast, the parallel architecture allows for independent training of the data-based model, potentially retaining the disadvantages of purely data-based approaches due to the lack of guidance from theoretical knowledge [14]. Figure 2.1 shows a general structure of a hybrid model as described in J. Pinto et al.(2022) [36].

The development of hybrid models from their birth to today reflects a more significant trend in scientific research towards more integrative and adaptable techniques. This progress has been fuelled by the growing complexity of the systems under investigation, as well as the demand for models that can effectively capture this complexity [39].

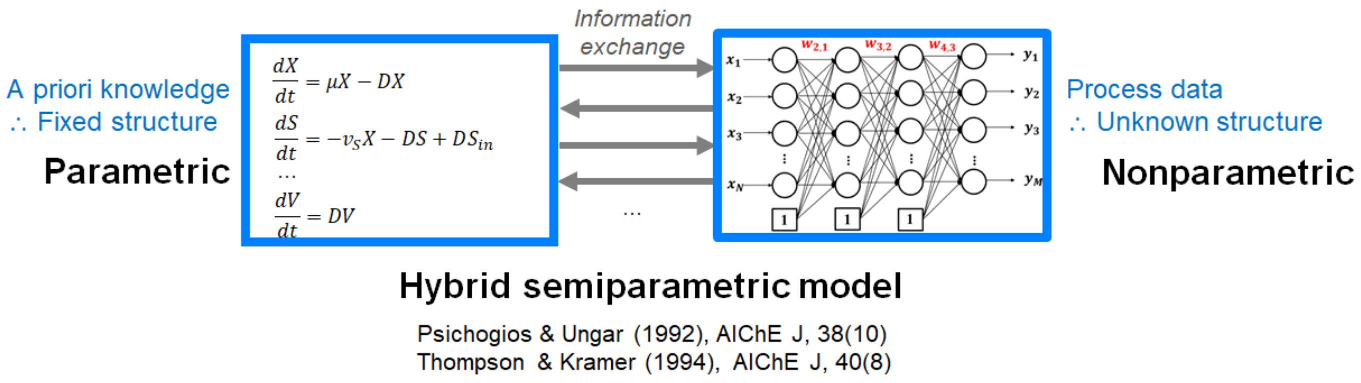

A priori knowledge
∴ Fixed structure
Parametric
$\frac{dX}{dt} = \mu X - DX$
$\frac{dS}{dt} = -v_S X - DS + DS_{in}$
$\frac{dV}{dt} = DV$
Information exchange
Process data
∴ Unknown structure
Nonparametric
Hybrid semiparametric model
Psichogios & Ungar (1992), AIChE J, 38(10)
Thompson & Kramer (1994), AIChE J, 40(8)

Figure 2.1: Schematic representation of the general deep hybrid model for bioreactor systems. Observable outputs (y) and a state vector (x) are components of the dynamic model. Due to the lack of explanatory mechanisms, it contains two components: a nonparametric component modeled by a deep feedforward neural network with numerous hidden layers and a parametric component with fixed mathematical structure (functions f(.) and h(.)) based on First Principles. The nonparametric component was found using process data. Adapted from [36].

2.3 Hybrid Models in Bioreactors

2.3.1 Parametric Models in Hybrid Systems

Parametric models serve as the backbone of hybrid systems, especially within the context of bioreactors. These models provide a detailed quantitative foundation essential for simulating, predicting, and optimizing bioreactor systems' complex interactions and responses under various operational conditions.

2.3.1.1 Foundation of Parametric Models

Parametric models are characterized by their reliance on parameters derived from both empirical data and theoretical insights into the biological and engineering processes governing bioreactor dynamics. This dual basis enables a comprehensive understanding of bioreactor behavior, facilitating the fine-tuning of operational parameters to enhance productivity and stability. The construction of these models typically involves several core elements:

- Mass Balance Equations: the mass balance equations describe the conservation of mass for the essential components engaged in the bioreactor system and offer a mechanism to investigate the ultimate destiny of influent nutrients. The substrate(s) being ingested, the biomass being generated or consumed, and any additional pertinent elements are all included in this. The model's core equations, known as the mass balance equations, are based on the rates of the reaction and transport activities taking place inside the bioreactor [40].

- Reaction Kinetics: in a bioreactor, the rates at which biological processes take place are described by the reaction kinetics. This covers the use of substrates, biomass formation, and any other pertinent reactions [41]. Different methods can be used to simulate the kinetics, including Michaelis-Menten kinetics [42], Monod kinetics [43], and more sophisticated models based on enzyme kinetics [44].

- Transport Phenomena: the transport processes that take place in bioreactors include the mass transfer of substrates, oxygen, or other gases, the transmission of heat, and the mixing of substances. These processes substantially impact the effectiveness of the bioreactor [45].

- Environmental Conditions: the model needs to consider the bioreactor's environmental factors, including temperature, pH, dissolved oxygen levels, and nutrient concentrations. These characteristics impact the growth and metabolic activity of biological organisms. It is possible to include feedback control methods to keep these circumstances within a desired range [46].

- Simulation & Optimization: simulation and optimization studies can be performed using the mathematical model after it has been created and validated. While optimization techniques can be used to identify the best operating settings or to optimize the bioreactor design, simulation enables prediction of the behavior of the bioreactor under various conditions [47].

2.3.1.2 Types of Parametric Models

Within the domain of parametric modeling, several types of models are employed, each offering different levels of detail and complexity:

- Monod Models: these are the simplest type of parametric models. They describe the growth of microorganisms in a bioreactor based on the concentration of a limiting substrate [48].

- Structured Models: take into account the cellular architecture. They can discuss the numerous metabolic processes that take place within distinct cell compartments, such as the cytoplasm, cell membrane, and nucleus [49].

- Unstructured Models: these models do not consider the internal structure of the cells. They treat the cell as a 'black box' and focus on the overall inputs and outputs of the system [49].

- Stochastic Models: consider the random nature of biological processes(i.e., randomness induced by the births and deaths in a large population) [50]. They are often used when the number of cells or molecules in the system is small, and random fluctuations can significantly impact the system's behavior.

The choice of model depends on the particular application and the level of information required. Each model has merits and disadvantages. Monod models, for instance, are straightforward to use, but they might not adequately depict the behavior of intricate bioreactor systems. Structured models, conversely, can offer a more thorough and precise description of the system, but they are more challenging to use and demand more computing power.

Parametric models can be used as digital twins for optimal design of experiments or process monitoring [51]. The number of modalities and varieties within the same modality is increasing, and a standardized workflow for carrying out these steps is required to accelerate the pipeline while reducing resources. A standardized protocol is necessary to capture the overall patterns in the data [2]. Parametric models can be highly complex, especially for large-scale industrial bioreactors and those involving multiple reactions and species. However, they provide a powerful tool for understanding and optimizing bioreactor performance.

In general, parametric models are pivotal tools in developing and optimizing bioreactor systems. By integrating detailed mechanistic understanding with empirical data, these models enable precise control and prediction of bioreactor performance, supporting the advancement of bioprocessing technology. Their application ranges from designing optimal experimental setups to serving as digital twins for real-time process monitoring and control, highlighting their critical role in enhancing biomanufacturing processes' efficiency and output quality.

2.3.2 Nonparametric models

Nonparametric models, intense learning models, represent a class of machine learning (ML) approaches that excel in identifying complex patterns within vast and nonlinear datasets. These models do not assume a predefined form for the relationship between input and output variables, making them exceptionally suited for applications in bioreactor technology, where such relationships are often intricate and poorly understood.

Deep learning models, a subset of nonparametric models, leverage artificial neural networks with multiple hidden layers to learn from data. This structure allows them to approximate functions that can capture the nonlinear dynamics of bioprocesses based on a wide array of inputs such as temperature, pH, dissolved oxygen, and nutrient concentrations. Consequently, they can predict key performance indicators—biomass growth, product concentration, and nutrient uptake—with notable accuracy, guiding the optimization of bioreactor conditions to enhance productivity and cost-effectiveness.

Remarkably, bioprocesses' modeling and optimization have extensively used deep learning models. Parametric models are frequently coupled with these models to create hybrid semiparametric systems [52]. To train these models traditionally, techniques like the Levenberg-Marquard method, cross-validation, and indirect sensitivity equations are used. Deep networks, also known as neural networks with numerous hidden layers, can, however, provide substantial advantages over their shallow equivalents, as recent developments in deep learning have shown [53, 54].

Deep networks are capable of approximating compositional functions with exponentially fewer parameters and sample complexity when compared with shallow networks. Additionally, they are less prone to overfitting, a typical issue in machine learning. The advent of stochastic gradient descent training techniques, specifically the ADAM approach [54], has aided the transition from shallow to deep network designs. Adaptive estimates of lower-order moments are the foundation of ADAM, a first-order gradient-based technique for stochastic objective functions. A straightforward and reliable training strategy that is less sensitive to gradient attenuation and the convergence to local optima was produced by combining the data subsampling with the learning rate adaption at each iteration. Since stochastic regularization based on weight dropout has successfully prevented overfitting in deep learning [55], it is widely used in conjunction with stochastic gradient descent methods.

2.3.3 Current Challenges in Hybrid Modelling

Hybrid modeling can tackle previously unsolvable problems using single-method techniques. However, integrating several modeling methodologies brings new issues that must be addressed to fully realize these systems' promise. These problems encompass technological, computational, and domain-specific issues that complicate the development and deployment of hybrid models [56].

1. Dealing with incomplete data. This circumstance complicates the use of ML to find patterns or correlations, as well as the use of multiscale models to anticipate system dynamics using well-known physics. The synergy between these approaches strives to overcome data restrictions, permitting the research of biological, biomedical, and behavioral systems with inadequate knowledge of their underlying physics [57].

2. The complexity of understanding systems whose physics isn't fully understood. This challenge is particularly noticeable when the data is incomplete and the physics underlying the systems has yet to be fully understood. Integrating ML and multiscale modeling in such contexts aims to leverage the strengths of both approaches, using ML to identify patterns and correlations in data and multiscale modeling to infer system dynamics from established physical laws despite the incomplete understanding of all underlying processes [57].

3. Balance between mechanistic knowledge and ML. Combining the deep, often qualitative insights from domain-specific theories with machine learning's quantitative, pattern-finding abilities. This integration is difficult due to the varied natures of the information processed by each approach. Mechanistic models rely on established physical rules and theories, but machine learning models excel at recognizing patterns in data without having to comprehend the underlying physics. Achieving

a good balance requires careful evaluation of how each method's strengths might complement the other, increasing prediction accuracy and insight [56].

4. Model Reduction vs. Model Identification. This issue arises from the conflict between simplifying complex models to make them computationally manageable (model reduction) and obtaining the correct model parameters from data (model identification). Achieving an appropriate balance entails ensuring that the simplified model still accurately depicts the system's basic dynamics, while efficiently identifying and estimating parameters. This effort is hampered by the intricate nature of many biological, chemical, and physical systems [56].

2.4 Deep Hybrid Models in Bioreactors

Hybrid semiparametric modeling has been used with deep learning approaches in bioreactor technology. For instance, deep feedforward neural networks with varying depths, the rectified linear unit (ReLU) activation function, dropout regularization of network weights, and stochastic training with the ADAM method have been explored. As mentioned in the publication by Pinto J et al. (2022) [36], these techniques have been used in case studies and compared to the conventional shallow hybrid modeling approach. The findings suggest that deep hybrid models' generalization abilities are systematically superior to shallow hybrid models. Furthermore, it is demonstrated that the deep hybrid models require less CPU time to train than their shallow counterparts.

2.4.1 General steps of building hybrid models

Building deep hybrid models in bioreactors involves several key stages, each critical to ensuring the model's accuracy and effectiveness. These steps are detailed below:

1. **Data Collection and Preprocessing:** after the experimental data is gathered, there may or may not be a preprocessing step to ensure no missing values or outliers.

2. **Mechanistic Model Development:** a mechanistic model based on theoretical principles is developed. The mechanistic model provides a foundation that captures the essential dynamics of the system, incorporating known physical and biological laws.

3. **Machine Learning Model Design:** in parallel, a ML model is designed. In the case of a deep learning model, this involves selecting the network architecture (e.g., number of layers, type of layers), activation functions (such as ReLU), and regularization techniques (such as dropout). After that, the optimizer is also chosen; ADAM is an example of a commonly used optimizer for training, leveraging its adaptive learning rate capabilities to handle large datasets efficiently.

4. **Hybrid Model Integration:** The core of the hybrid modeling approach is the integration of the mechanistic and machine learning models. This can be achieved using a serial or parallel architecture [2]:

   - **Serial Architecture:** The outputs of the mechanistic model are used as inputs or constraints for the machine learning model. This sequential integration allows the empirical data to refine and enhance the predictions made by the mechanistic model.
   - **Parallel Architecture:** Both models operate simultaneously, combining their outputs to form a comprehensive prediction. This approach leverages the strengths of both models, offering a robust representation of the bioreactor system.

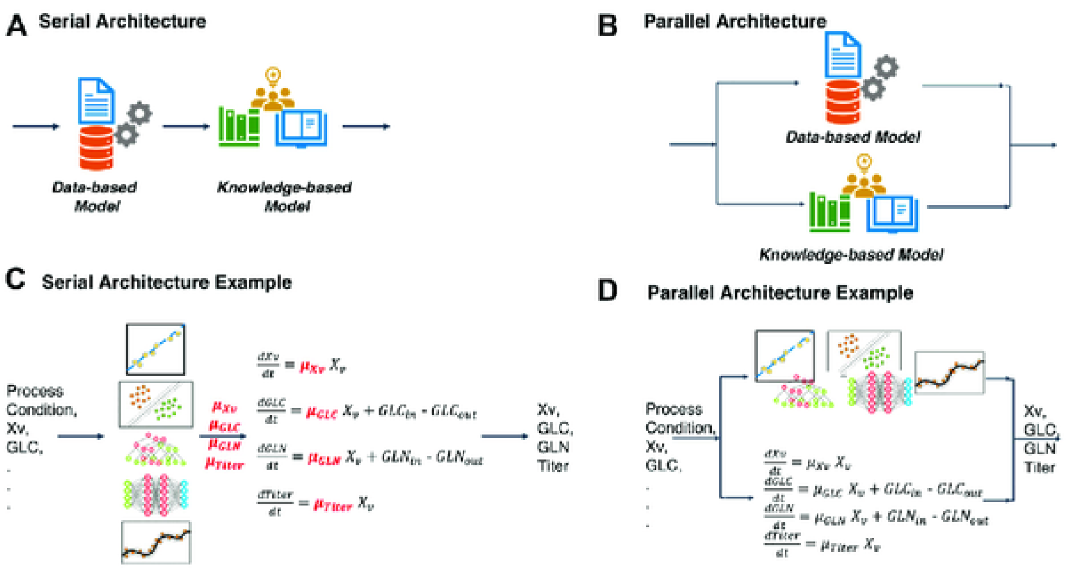

A Serial Architecture
Data-based Model
Knowledge-based Model
B Parallel Architecture
Data-based Model
Knowledge-based Model
C Serial Architecture Example
Process Condition, Xv, GLC,
μ_{Xv} μ_{GLC} μ_{GLN} μ_{Titer}
$\frac{dXv}{dt} = \mu_{Xv} X_v$
$\frac{dGLC}{dt} = \mu_{GLC} X_v + GLC_{in} - GLC_{out}$
$\frac{dGLN}{dt} = \mu_{GLN} X_v + GLN_{in} - GLN_{out}$
$\frac{dTiter}{dt} = \mu_{Titer} X_v$
Xv, GLC, GLN Titer
D Parallel Architecture Example
Process Condition, Xv, GLC,
Xv, GLC, GLN Titer

Figure 2.2: Schematic representation of different architectures of hybrid models, (A) serial and (B) parallel architecture, illustrated with an example (C)-Serial architecture, (D)-Parallel architecture. From [2]

5. **Training:** the hybrid model is trained using observed process data. During training, the model parameters are adjusted to minimize prediction errors. This stage involves refining the model by evaluating its performance on validation datasets and adjusting the architecture or parameters as necessary.

6. **Implementation and Monitoring:** the final stage involves implementing the hybrid model in a real-time bioreactor control system. Continuous monitoring and periodic re-evaluation of the model's performance are crucial to account for any changes in the bioreactor dynamics. The model may be updated regularly with new data to maintain accuracy and reliability.

By following these stages, process researchers and engineers can develop hybrid models that effectively capture the complex dynamics of bioreactors, leading to improved prediction accuracy and enhanced (bio)process optimization.

3 Related Work

This section will explore available hybrid modelling tools and how they currently impact the bioprocess field.

3.1 Existing tools

Even though using hybrid models is appealing, building such models involves several challenges. To facilitate and guide the workflow/pipeline from process data upload (inputs) to hybrid model deployment towards generating model-derived insights of bioprocesses, only a few efforts have been attended.

3.1.1 Commercial Software

In the landscape of hybrid modelling, several commercial software tools have emerged, offering robust solutions for integrating mechanistic and machine learning approaches. Two notable examples are Aspen Hybrid Models [58] and Novasign Toolbox [59]. These platforms provide advanced functionalities tailored for the complexities of hybrid modelling, catering to various industrial and research applications.

Aspen Hybrid Models, accessible at Aspen Hybrid Models, stands out for its comprehensive suite of tools designed to streamline the development of hybrid models. It facilitates the seamless integration of process simulations with ML algorithms, enabling users to enhance predictive accuracy and optimise process design and operations. However, it is not freely available as a commercial product, which may limit its accessibility for academic research and small-scale applications.

Similarly, Novasign Toolbox, available at Novasign Toolbox offers a specialised platform for developing and deploying hybrid models, particularly in the realm of biopharmaceutical processes. It combines data-driven modelling capabilities with process understanding, improving process development and manufacturing decision-making. Like Aspen Hybrid Models, Novasign Toolbox is a proprietary solution requiring a licence, which may pose a barrier to widespread adoption.

3.1.2 HybridML platform

More recently, a flexible alternative to the commercial options has been launched (a toolbox called HybridML [0]); here, the training of the hybrid models employs Tensorflow [60] for training the neural networks and Casadi [61] for the integration of ODEs. The HybridML platform makes it easier to create and use hybrid models. It makes it simple to manage and generate models for hyperparameter search in distributed situations by allowing users to declare models in a single file using a JSON interface. Nevertheless, this framework requires the user to have some background knowledge of hybrid model development and coding languages.

3.1.3 SBML2HYB tool

SBML2HYB [62], is a python tool intended as an interface to convert existing SBML models into a hybrid model (combines mechanistic equations and ML techniques). The SBML2HYB tool is implemented in Python 3 and uses libSBML and the xml.etree submodule to translate the SBML files to `.hmod` (intermediate format — enables communication between the essential components of the mechanistic and hybrid models) and vice versa. To validate `.hmod` files, a new syntax validator was built, and two Python modules were constructed: one that receives an SBML, validates the file, and builds an `.hmod` file, and validates it; the other module transforms an `.hmod` file into an SBML, which is also validated[62]. The SBML2HYB pipeline was demonstrated on two case models: the threonine synthesis pathway model of Escherichia coli [63] and the Park and Ramirez [64] model. First, using the SBML2HYB tool, the SBML files were transformed into `.hmod` files.

The ML component information was then included. The parameters and assignments related to ML could be written back into the `.hmod` file because the `.hmod` format contains data from the SBML and ML. The SBML file with the implemented hybrid model was then created by translating the obtained `.hmod` file using the SBML2HYB tool. The results of uploading these SBML files back into the BioModels database showed that the hybrid models had been successfully trained and converted into an SBML format because they were very similar to the original mechanistic models [62]. However, this tool does not include the training/validation step.

3.2 MATLAB version tool

The in-house hybrid modeling tool developed over the years in MATLAB is a significant resource that has been used in a substantial number of studies by our research group. This MATLAB version is used in a closed setting, catering to specific case studies and research efforts within the organization. A notable application of this MATLAB tool is highlighted in a case study. This case study demonstrates the tool's capacity to integrate complicated biological processes with computational modelling to produce useful information, underscoring its importance in bioprocess research and development. The significance of the in-house MATLAB tool is a source of pride for our research group and a testament to our commitment to advancing hybrid modelling in the bioprocess field.

The MATLAB application marks a significant step forward in our research team's hybrid modelling approaches. It has permitted a more profound understanding of bioprocess dynamics and established a methodological foundation for the Python tool.

Overview of related Tools The table below concisely presents an overview of the related tools:

Table 3.1: Comparison of Hybrid Modeling Tools.

Tool	Development Language	Sofware Used	Availability	Main Features	Reference
Aspen Hybrid Models	Not Public	AspenTech	Commercial	Create and Train Hybrid Models	[58]
Novasign Toolbox	Not Public	Novasign	Commercial	Create, Train and Evaluate Hybrid Models	[59]
HybridML	Python	Python, TensorFlow, Casadi	Open-source, requires Python expertise	Create and Train Hybrid Models	[0]
MATLAB version tool	MATLAB	MATLAB, Symbolic Math Toolbox	In-house	Create, Train and Evaluate Hybrid Models	[36]

4 Methodology and Solution

This chapter describes the approach and solution to accomplish the objectives outlined in Chapter 1. It represents the stages taken in the development, the main parts of the HYBpy framework and website, and the reasoning behind the choice of tools and libraries used, including some that were taken into consideration but were not used in the final version.

4.1 HMOD model information gathering

This first section discusses the approach for extracting and processing information from the `.hmod` file, which is required for initializing and configuring the HYBpy pipeline. The internal HMOD is an intermediate format that enables the communication between the essential components of the mechanistic and hybrid models described in [29] (see an example file in https://github.com/joko1712/HYBpy_ModelsandData). There are two ways of generating this file:

1. If the user posses a SBML model, the tool SBML2HYB can be used to generate the `.hmod` file from that model. After extracting the `.hmod` file from the SBML2HYB programme (which uses Python to convert SBML files to `.hmod` and converts systems biology mechanistic models), the `.hmod` file is imported into HYBpy.

2. The user has also the option of creating the `.hmod` file from scratch. This is the approach we used when generating one of our case studies that will be analyzed in Chapter 5.

On both occasions the model has the possibility of not having any information about the machine learning component, if such the HYBpy tool will notify the user and will guide them so that this component can be created for the training of user model.

The *hybdata* Python script is designed to automate the process of loading the `.hmod` file, assuring efficiency and accuracy in managing the different information included within the `.hmod` file and withdraw the relevant information.

4.1.1 Purpose of the *hybdata* script

The .hmod file, which contains the descriptive details of hybrid models, including their parameters, structures, and dynamics, serves as the blueprint for constructing and simulating these models. However, the raw format of `.hmod` file, while comprehensive, is not immediately conducive to computational processing or model initialization due to its complexity and diversity in the structure. This is where the *hybdata* script comes into play.

Acting as a translator, the *hybdata* script meticulously parses the `.hmod` files to extract and organize the critical information needed to initialize and configure hybrid models within the HYBpy framework. It transforms the static data contained within `.hmod` files into a dynamic and structured format that can be directly utilized by the HYBpy framework to create and simulate the hybrid models. The data is stored in a created dictionary. The figure 4.1 better demonstrates this progress.

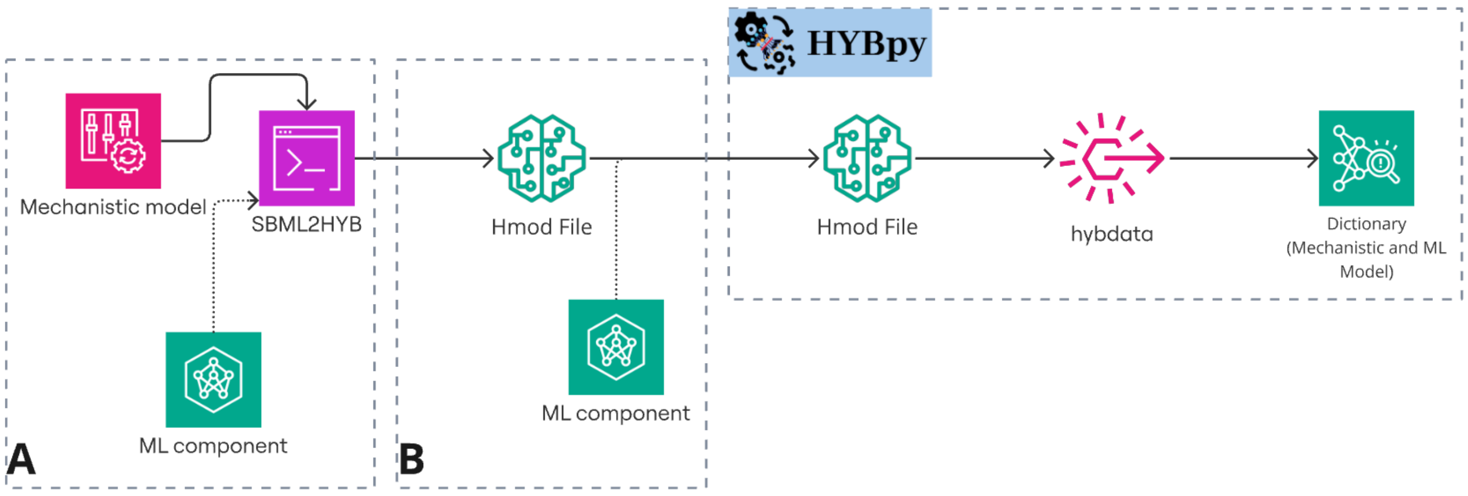


Figure 4.1: Illustration of how the *hybdata* script processes the `.hmod` model format, detailing the import and processing within HYBpy. The `.hmod` file can incorporate machine learning settings either directly from the SBML2HYB tool or manually by the user, with the assistance of the HYBpy tool, when starting with a mechanistic-only `.hmod` file. This diagram demonstrates both pathways for integrating machine learning components into the HYBpy framework through the *hybdata* script. Section A shows that a user can train a model using a `.hmod` file generated by SBML2HYB, adding the ML component at that stage or later. Section B illustrates that a user can start from an empty `.hmod` file, manually adding the mechanistic model (as was done for the *Basic* model) and the ML component, or the user can upload a `.hmod` file with a mechanistic model and add the ML component within the HYBpy interface. The *hybdata* script facilitates the parsing and integration of these components within HYBpy, enabling efficient model initialization and simulation.

4.1.2 *hybdata* steps overview

The *hybdata* script executes a series of steps to convert the contents of `.hmod` model files into a structured format usable by the HYBpy framework.

With the file contents loaded, *hybdata* proceeds to systematically parse the various sections defined within the `.hmod` file. This involves recognizing and extracting information related to:

- Species: identifying all biological entities involved in the model, including their unique identifiers (IDs) and initial concentrations.
- Compartments: defining the compartments.
- Parameters: extracting model parameters.
- Rules and variables: identifying rules for assignments, mathematical equations, and variables that might change over the course of a simulation.
- Machine learning block: specifying all necessary information about the machine learning algorithm and neural network model used for training. This includes the network architecture, activation functions, training parameters, and hyperparameters. If the `.hmod` file does not contain ML information, this block is added by the user through the web-based interface.

Upon completing the parsing process of the hybrid model, *hybdata* stores the extracted information in structured dictionaries. This facilitates easy access to model components during the simulation phase and allows for modifications and updates to the model without having to revisit the raw `.hmod` file.

4.2 CSV data information gathering

This section delineates the methodology employed for extracting, processing, and labeling the experimental data from `.csv` files, specifically aimed at enhancing the dataset utilized within the HYBpy framework. The script detailed here, `csv2json`, distinct yet complementary to the *hybdata* script, focuses on the structured extraction of observed time-series species data for different batch (runs) from the `.csv` files, facilitating the subsequent integration of this data with the information obtained from the `.hmod` files to train the models.

4.2.1 Steps Overview

The script begins by loading the project configuration from the dictionary, which contains parameters and settings previously extracted from an `.hmod` file. It then proceeds to read the `.csv` file, for example, chassbatch1.csv, which contains multiple batches with various time points. This data is crucial for training and testing the hybrid models, providing real-world or simulated inputs that complement the model structure, defined by the `.hmod` files.

The parsing process involves several key steps:

- Reading CSV headers: the script identifies the headers in the `.csv` file (see data file structure template in HYBpy), which denotes in which column is "time", species measured concentration and their standard deviation captured at each time point across various batches (each line represent a new time value).
- Grouping data by time and batches: Data points are grouped according to their batch and within each batch by their time points.
- Handling empty rows: empty rows, which might signify the transition between different batches, trigger the creation of a new batch group, ensuring that each batch's data is kept distinct (this will only occur if the whole line is empty).

After the batches are organized, we will label batches for training and testing purposes, a step critical for the model's learning and validation phases. The script offers two modes for batch selection:

- Manual labeling: users can manually specify which batches should be used for training and which for testing, allowing for targeted experiments and analysis.
- Random labeling: alternatively, batches can be randomly assigned to training and testing sets, facilitating unbiased model evaluation.

The script further processes the data by:

- Adding state and time information: For each time point within a batch, the script calculates and stores the state vector, representing the system's condition.
- Calculating summary metrics: Additional metrics, such as the number of time points per batch and the overall number of batches, are computed and added to the dataset, providing valuable insights for model simulation and analysis.

4.3 Settings selection (*hytrain init*)

The `.hmod` is crucial for this step because all the information regarding the models and how to create the neural network is inside it. The *hybtrain* follows a straightforward structure:

1. Training mode identification - the model can be trained with three training methods:
    - Indirect mode, the residual, and Jacobian calculations are segregated from the system's forward simulation. Typically, this mode employs the derivatives of the states concerning the model parameters computed through a solver (e.g., an ODE solver).
    - Direct mode, residual, and Jacobian calculations are fully integrated into the neural network's forward pass. This strategy often relies on automatic differentiation provided by deep learning frameworks.

- Semidirect mode, the calculation of residual and Jacobian is closely linked to the forward simulation. This usually means utilizing the ANN directly to compute the output derivatives concerning the weights during the forward pass.

Continuing from the training mode identification, the subsequent steps in the *-hybtrain* initialization and execution process further detail how the hybrid training framework operates, emphasizing the integration of ML models with mechanistic insights for optimizing bioprocess simulations.

2. Parameter and Hyperparameter Setup - after determining the training mode, the framework initializes the model parameters and hyperparameters. This includes setting up network architecture specifics, such as the number of hidden layers, neurons per layer, and activation functions, based on insights extracted from the `.hmod` file. Additionally, training hyperparameters like learning rate and the number of training iterations are defined.
3. Model Compilation - With the training mode selected and parameters set, the model is compiled. This involves initializing the neural network with the specified architecture and preparing it for training. The compilation process also includes setting up the loss functions and optimization algorithms to be used during the training process.
4. Training Execution - The compiled model then undergoes training. This process adjusts the model parameters to minimize the difference between predicted and observed outcomes according to the selected training mode. The framework dynamically adjusts the training procedure based on performance metrics.
5. Deployment and Integration - Finally, the trained and validated model is prepared and inserted into the `.hmod` file. This step ensures the trained model is integrated into the `.hmod` file and seamlessly with existing systems or processes, facilitating real-world applications.

4.4 Neural Network Explanation

The Neural Network (NN) architecture in the HYBpy framework is a versatile and critical component designed to model complex relationships within the bioprocess data. Here's an overview focusing on how the NN is created.

4.4.1 Custom MLP Network Creation

The *mlpnetcreate* function initializes the CustomMLP neural network using the parameters specified in the *projhyb* configuration object, such as the number of input and output nodes (ninp, nout), the number and size of hidden layers (NH), and the types of neurons to be employed (neuron). The network's flexibility is demonstrated by its capacity to modify the activation functions used in the hidden layers, allowing it to adapt to diverse modeling circumstances.

4.4.2 Neural Network Layers

The CustomMLP class extends PyTorch's nn.Module, allowing users to build a multi-layer perceptron with arbitrary layer types. Each layer type—TanhLayer, ReLULayer, and LSTMLayer—implements a unique neural activation function, allowing the model to detect non-linearities and complicated patterns in the data.

- **TanhLayer:** The hyperbolic tangent activation function, as described in various research papers, e.g., [65], is effective in neural networks for its ability to center data around zero. This characteristic is beneficial for optimization because it ensures that the average output of the neurons is closer to zero, making the network less likely to be stuck in regions with zero gradients.

- **ReLULayer:** The ReLU activation function has become the default activation function for many types of neural networks. Its simplicity and effectiveness in avoiding the vanishing gradient problem make it ideal for deep networks. ReLU activates neurons only if the input is positive, introducing sparsity into the network, which can lead to more efficient computations [65].

- **LSTMLayer:** The LSTM architecture, explained comprehensively by Christopher Olah [66], addresses the limitations of traditional RNNs by introducing a memory cell that can maintain its state over long periods. This is achieved through its gating mechanisms, which control the flow of information into and out of the cell. This capability makes LSTMs particularly powerful for tasks involving sequences, such as time-series prediction and natural language processing.

4.4.3 Weight Initialization and Scaling

Upon network creation, weights are initialized using strategies appropriate for each layer type (e.g., Xavier/Glorot uniform initialization for TanhLayer and Kaiming/He uniform for ReLULayer, presented in figure 4.3) to promote effective learning. The scale_weights method refines this initialization by applying a scaling factor, enhancing the network's sensitivity and convergence properties during training.

4.4.4 CustomMLP

Here's a detailed explanation of the CustomMLP class and its methods:

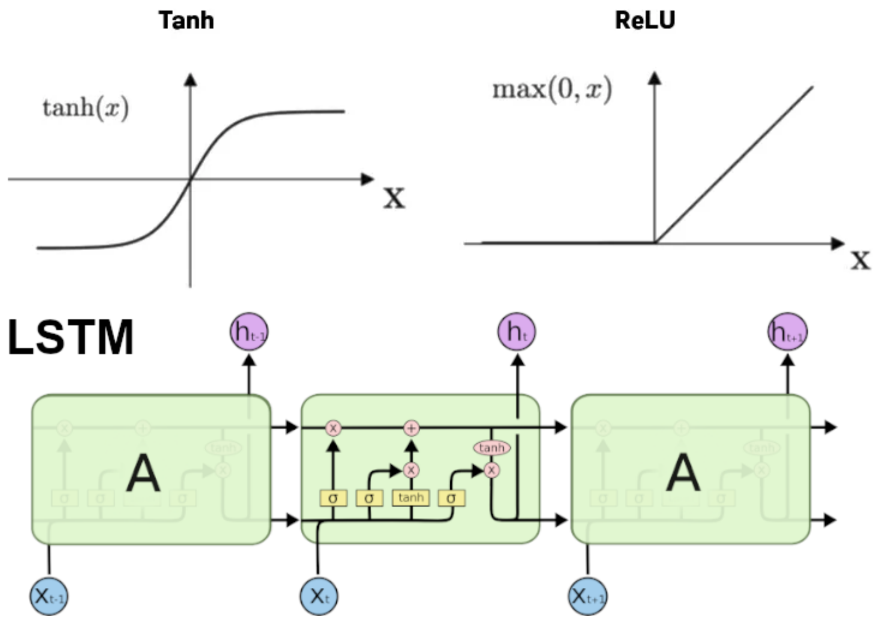


Figure 4.2: Diagram of the different Activation Functions. Adapted from: [67] [66].

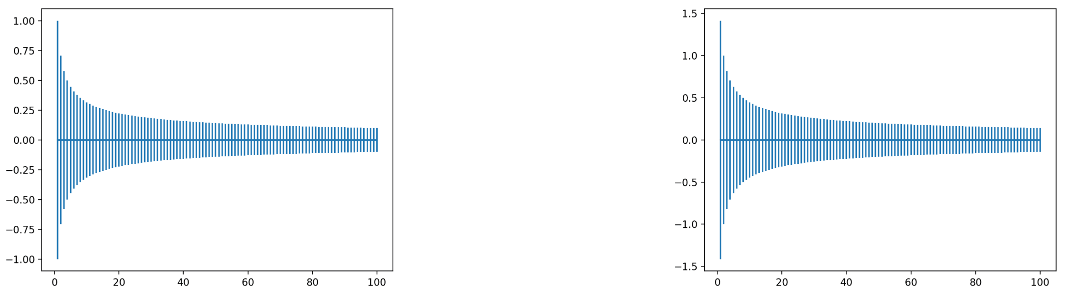


Figure 4.3: On the left, a plot of Range of Xavier Weight Initialization, on the right, a plot of Range of Kaiming Weight Initialization, both with inputs from one to one hundred. Taken from [68]

- **Initialization** The init method initializes the layers according to the layer types (tanh, relu, lstm). It uses a ModuleList to store these layers, making it simple to go through them during forward propagation.

- **Forward Propagation** The forward method determines how input data is sent through the network. Each layer applies its activation function on the incoming data, altering it at each stage until the final prediction is made. Converting the input to torch.float64 ensures numerical stability during computations.

- **Backwards Propagation** The backpropagate method aims to compute the gradients of the loss function concerning the network weights, which is required throughout the training phase. This method uses the backpropagation algorithm, an essential part of neural network training.

- **Layer Classes** Each activation layer will have a different class to deal with the data. These are the classes available:

  1. TanhLayer: This class implements a layer with the hyperbolic tangent activation function. It initializes weights and biases using Xavier uniform initialization, ensuring effective learning.
  2. ReLULayer: This class implements a layer with the ReLU activation function, using Kaiming initialization to effectively handle the problem of vanishing gradients.
  3. LSTMLayer: This class implements a layer with LSTM units designed to handle sequences and maintain information over long periods, making them suitable for tasks like time-series prediction.

4.4.5 Learning Methods

The *hybtrain* function in the HYBpy framework represents the many optimization strategies used to train the neural network. The chosen method will rely on the information in the `.hmod` file. The learning methods implemented in HYBpy are:

1. **Method 'Trust Region Reflective (trf)'**: The method 'trf' (Trust Region Reflective) is inspired by the process of solving a system of equations that provide the first-order optimality condition for a bound-constrained minimization problem as defined in [30]. The algorithm solves trust-region subproblems iteratively, using a particular diagonal quadratic term and a trust-region shape determined by the distance from the boundaries and gradient direction. This improvement allows you to avoid taking steps directly into bounds and instead explore the entire space of variables.

2. **Method 'L-BFGS-B':** Byrd, R. H., P. Lu, and J. Nocedal introduced 'L-BFGS-B,' a limited-memory algorithm for bound-constrained optimization, in 1995 [31]. It is especially useful for large-scale problems because it utilizes a small amount of memory. The approach approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm with limited computer memory [69]. This method is particularly useful for issues with simple box restrictions and has been widely used in various scientific and technical applications.

3. **Method 'trust-constr':** The 'trust-constr' method implements a trust-region technique for constrained optimization. It changes between two implementations based on the problem definition. It implements the Byrd-Omojokun Trust-Region SQP method for problems with equality constraints [32]. When inequality constraints are imposed, it uses the trust-region interior point approach [70]. This interior point technique uses slack variables to solve a series of equality-constrained barrier problems for progressively decreasing values of the barrier parameter. The

equality-constrained SQP method solves subproblems with increasing precision as the iteration progresses towards a solution. This makes the strategy more adaptable and appropriate for large-scale problems.

4. **Method 'dual_annealing':** This function performs the Dual Annealing optimisation. This stochastic approach, derived from [33], combines the generalization of CSA (Classical Simulated Annealing) and FSA (Fast Simulated Annealing) [71] [72] and a strategy for applying a local search on accepted locations [73]. Y. Xiang et al. [74] describes an alternative implementation of this technique, whereas [75] presents benchmarks. This approach presents an advanced way of refining the generalized annealing solution.

5. **Method 'ADAM':** Diederik P. Kingma and Jimmy Ba presented the ADAM (Adaptive Moment Estimation) optimization technique in 2014 [76]. It incorporates the advantages of two existing stochastic gradient descent modifications. It determines adaptive learning rates for each parameter by estimating the first and second moments of the gradient. The approach updates parameters by taking the momentum of previous gradients (as in RMSProp [77]) and the moving average of squared gradients (in AdaGrad [78]). This makes ADAM ideal for situations with huge datasets and high-dimensional parameter spaces.

The methods 'trf', 'L-BFGS-B', 'trust-constr', 'dual_annealing' are imported via the Scipy package [79], from scipy.optimize.least_squares for: 'trf'; scipy.optimize.minimize for: 'L-BFGS-B' and 'trust-constr'; scipy.optimize.dual_annealing for: 'dual_annealing'. ADAM is imported via the Pytorch package [80], from torch.optim.Adam.

4.5 Objective Function and Jacobian

When creating a solid hybrid modeling tool, precisely handling the objective function and its derivatives, such as the Jacobian, is critical. Control and accuracy are the main arguments for using customized functions for these calculations rather than the default methods offered by optimizers. This section addresses the significance of custom objective functions and Jacobian calculations and provides a full explanation of how these functions are implemented in the provided code.

As mentioned, the reasons for the custom calculations are:

1. Control over the computations:

    - Custom functions enable precise control over the optimization process. For example, special constraints, unique regularisation terms, or domain-specific logic can be integrated directly into the objective function.
    - Optimizers' default functions frequently use generic implementations that may not properly match the specific requirements of complex hybrid models.

2. Improved Accuracy and Efficiency:

   - Calculating the objective function and Jacobian using well-defined numerical methods results in greater accuracy. This is especially important in hybrid models, where the relationship between different components (e.g., data-driven and physics-based models) requires accurate control.
   - Custom Jacobian calculations can greatly reduce computing load by eliminating the requirement for finite difference approximations, which are both computationally expensive and prone to numerical errors.

4.5.1 Implementation of Custom Objective Function and Jacobian Calculation

The *hybtrain* function integrates the custom objective function and Jacobian into the training process. The function orchestrates the optimization process, configuring parameters and settings based on user input (taken from the `.hmod` file).

First, the custom objective function measures the error between the model's predictions and the observed data. It considers hybrid models' specific structure, ensuring that mechanistic and data-driven components are correctly represented.

The Jacobian matrix, which represents the partial derivatives of the residuals concerning the weights, is critical for optimization techniques that use gradient information. The custom Jacobian calculation is designed to compute these derivatives efficiently using automatic differentiation techniques.

These custom functions are integrated into *hybtrain's* optimization methods. Depending on the method and mode chosen for the training, the proper configuration is established, and custom functions are supplied as arguments.

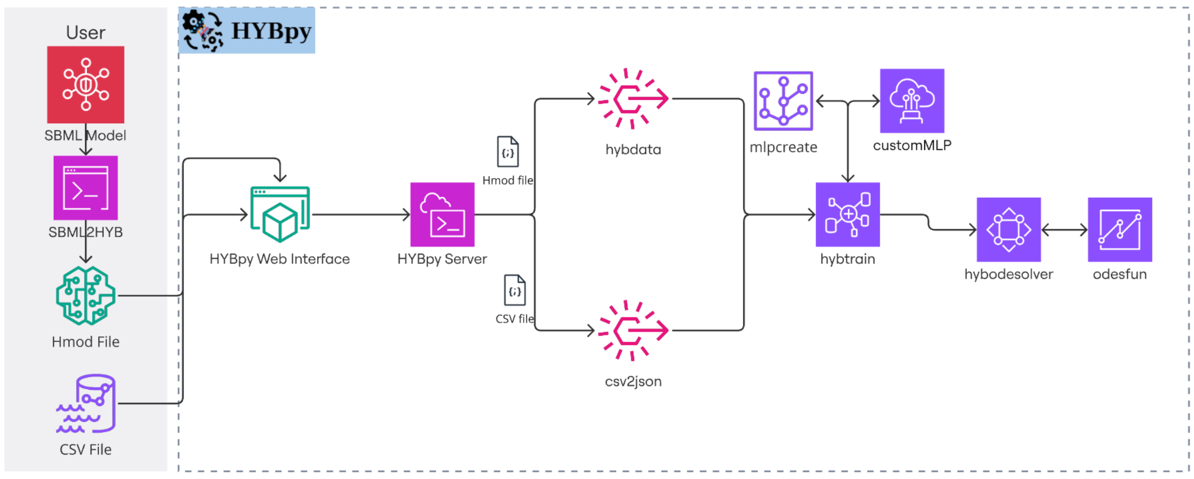


Figure 4.5: Architecture diagram of the HYBpy tool, highlighting the functional flow from initial model loading to simulation and optimization. The architecture allows the user to start with a SBML model or directly with an `.hmod` file, eliminating the need for prior conversion from an SBML file, thereby facilitating the integration and manipulation of hybrid and mechanistic models with machine learning components.

This concludes the methodology sections related to the tool's Python implementation. In the next sections, we will focus on the web service and graphical user interface aspects of the HYBpy platform.

All presented results and simulations were obtained through the HYBpy web-based platform, and the code was implemented in the Python programming language in a device with the following processor specifications: Processor AMD Ryzen 7 5800H with Radeon Graphics, 3201 Mhz, 8 Core(s), 16 Logical Processor(s). The files with the models and data can be accessed in: https://github.com/joko1712/HYBpy_ModelsandData

4.6 Transition from the MATLAB tool

As described in the section 3.2, this Python framework is a successor of an in-house developed tool in MATLAB. The transition from MATLAB to Python was motivated by a desire to enhance accessibility (i.e., not requires proprietary software to function), usability and extend the tool functionalities. By adopting Python, an open-source platform, the tool gains broader applicability and facilitates collaboration within the hybrid modeling community. Moreover, Python's extensive libraries and active development community offer substantial advantages in developing more sophisticated models and integrating machine learning algorithms more seamlessly.

The Python tool's advanced features and methodology were built on the foundation laid by the MATLAB tool. It provided essential insights into the prerequisites for efficient hybrid modeling, integrating several data sources, and the practical issues of bioprocess optimization. This foundational work has influenced the creation of the Python tool, ensuring that it not only retains the MATLAB version's strengths but also offers new capabilities that answer previously unmet demands.

4.7 Web Service and User Interfaces

The webapp interface for HYBpy is designed to provide a user-friendly experience for process researchers and practitioners, enabling them to interact with the framework without requiring extensive programming knowledge. It was developed using React [81] and Material-UI (MUI) [82], combined with Firebase [83] for authentication and user storage management, Flask [84] was used in the backend to handle requests and file processing.

4.7.1 Frontend (React)

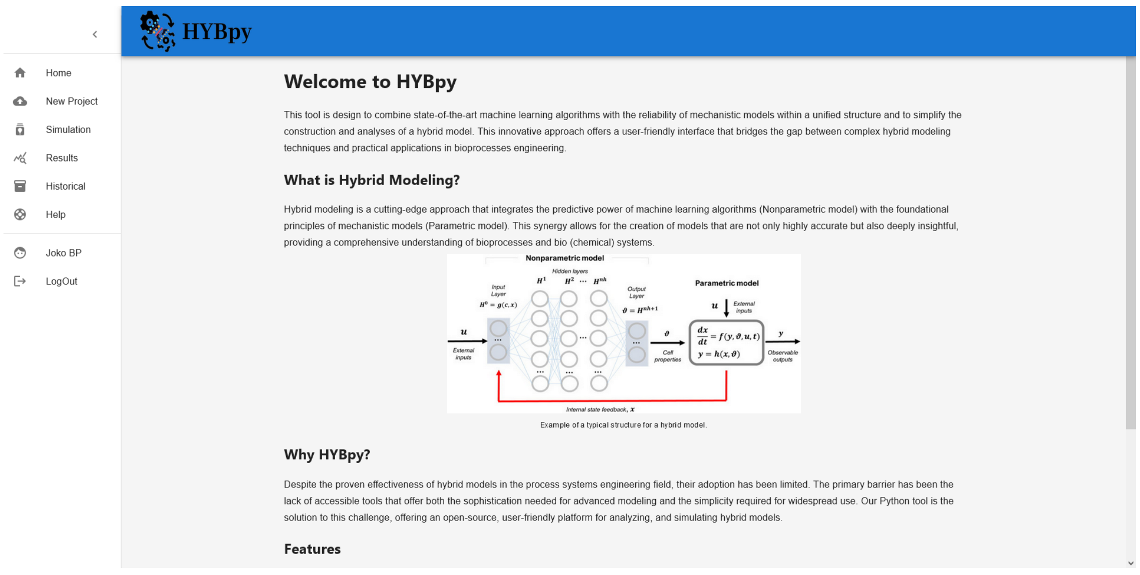
React is a popular JavaScript library for building user interfaces, known for its efficiency and flexibility. The HYBpy web-based platform uses React to create a responsive and intuitive interface that allows users to interact with the framework's features.

4.7.1.1 Main Components

The app uses Material-UI's AppBar, Drawer, and Toolbar to provide a responsive, consistent layout. The components from Material-UI are used to create a clean, modern interface that is easy to navigate. The main components are:

- **Registration/Login:** These pages allow users to create an account or log in to an existing one. Firebase Authentication is used to handle user registration and log-in.
- **Home:** Page where the user is presented with the main functionalities of the platform.
- **New Project:** Page where the user can initiate a model's training.
- **Simulations:** Page where the user can simulate a trained model with different data.
- **Results:** Page where the user can access the last model that was trained.
- **Historical:** Page where the user can view previously trained models.
- **Help:** Page where users can find a step-by-step guide for HYBpy.

The platform uses React Router [85] to handle the navigation between pages, ensuring a smooth user experience. React's useState and useEffect hooks handle dynamic updates like file uploads and API calls. State management ensures that all user interactions are reflected in the app's UI. The Home page can be observed in the screenshot presented in 4.6



Welcome to HYBpy

This tool is design to combine state-of-the-art machine learning algorithms with the reliability of mechanistic models within a unified structure and to simplify the construction and analyses of a hybrid model. This innovative approach offers a user-friendly interface that bridges the gap between complex hybrid modeling techniques and practical applications in bioprocesses engineering.

What is Hybrid Modeling?

Hybrid modeling is a cutting-edge approach that integrates the predictive power of machine learning algorithms (Nonparametric model) with the foundational principles of mechanistic models (Parametric model). This synergy allows for the creation of models that are not only highly accurate but also deeply insightful, providing a comprehensive understanding of bioprocesses and bio (chemical) systems.

Example of a typical structure for a hybrid model.

Why HYBpy?

Despite the proven effectiveness of hybrid models in the process systems engineering field, their adoption has been limited. The primary barrier has been the lack of accessible tools that offer both the sophistication needed for advanced modeling and the simplicity required for widespread use. Our Python tool is the solution to this challenge, offering an open-source, user-friendly platform for analyzing, and simulating hybrid models.

Features

Figure 4.6: Screenshot of the HYBpy Home page.

4.7.2 Backend (Flask)

The Flask backend handles requests from the frontend, processes files, and interfaces with the HYBpy framework. The files are stored temporarily before uploading them to Google Cloud Storage using Firebase. The backend is also responsible for using the Firebase Admin SDK to authenticate users, manage user-specific folders, and store the model training results. The backend provides several endpoints to handle the app's different functionalities.

The architecture of the HYBpy blends React for the frontend and Flask for the backend, providing a seamless user experience and efficient data processing. The frontend is designed to be intuitive and user-friendly, while the backend ensures that the app runs smoothly and securely. Combining these technologies enables HYBpy to deliver a powerful and accessible tool for hybrid modeling, catering to a wide range of users in the biosystems engineering community. The use of Firebase ensures scalable storage and data management.

5 Results and Discussion

5.1 HYBpy web-based platform

This section will explain the website's full interface and their components. HYBpy is available online at www.hybpy.org

5.1.1 Registration and Login

When users first open the website, they will be prompted to register or log in. In the future, there will be an option to use the website without an account; however, it will be communicated that this will limit their access to certain features and capabilities. Users can create an account using their email and a password or register directly using their Google account.

5.1.2 Home Page

After successfully logging in, users are directed to the Home page, which serves as the central hub for information regarding the tool. The home page introduces HYBpy as a user-friendly platform designed to simplify the integration of machine learning algorithms with mechanistic models. This hybrid modeling technique offers a powerful way to combine the flexibility of nonparametric models (machine learning) with the accuracy and interpretability of parametric models (mechanistic models), making it ideal for complex bioprocess engineering challenges.

5.1.3 New Project Page

The "New Project" page is a crucial part of the website where users can upload the necessary `.hmod` and `.csv` files to generate and train a model. Users also have the option to use pre-provided files available on the website. Initially, all options except for the title and the offered files will be grayed out. Only after the user assigns a title to the model can they upload the necessary files.

After uploading or selecting the `.hmod` file, users can edit machine learning-related information for the model. A button next to the upload button for the `.csv` file allows users to easily view graphs of the different batches presented in the `.csv` file.

Suppose the machine learning section is missing after uploading the `.hmod` file, relevant modals will pop up to prompt the user to fill in the necessary information. Once everything is in order, users can access or alter this information via the button next to the `.hmod` upload button.

Here is a detailed explanation of the process:

Title Assignment Users must first provide a title for the model. This step is essential as it enables the subsequent options for file uploads. The title field is a mandatory input that helps organize and identify the models within the user's workspace.

Uploading Files After providing a title, users can upload the necessary `.csv` and `.hmod` files. The `.csv` file contains the dataset with batch information, while the `.hmod` file contains the configuration for the hybrid model, including machine learning parameters and mechanistic model settings.

Viewing Batch Data Upon uploading the `.csv` file, users can click a button to visualize the data. This visualization presents graphs of the different batches, aiding users in understanding the structure and content of their data. This step ensures that users can verify their dataset before proceeding with model training. As seen in Figure 5.1

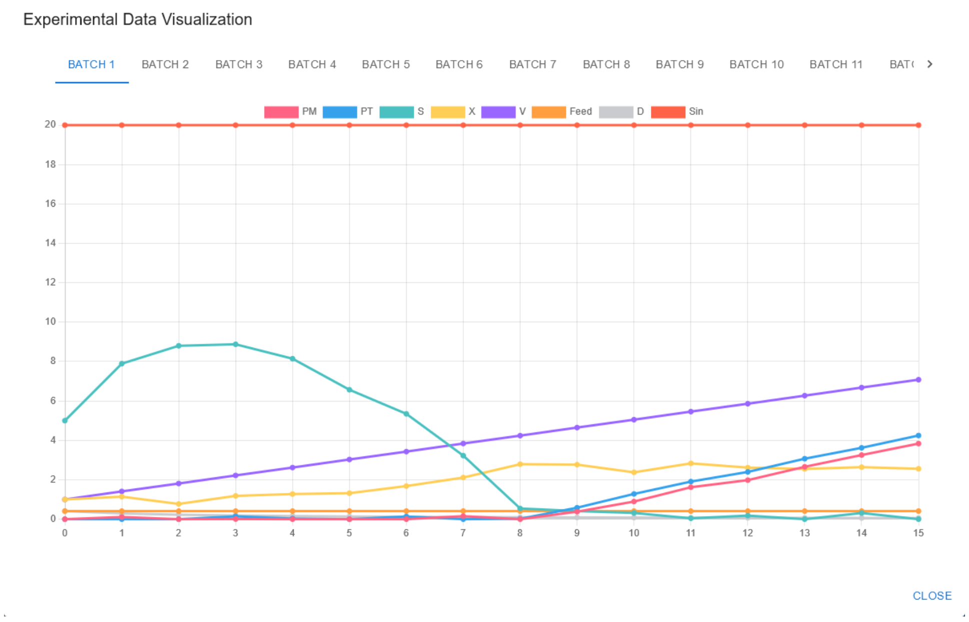
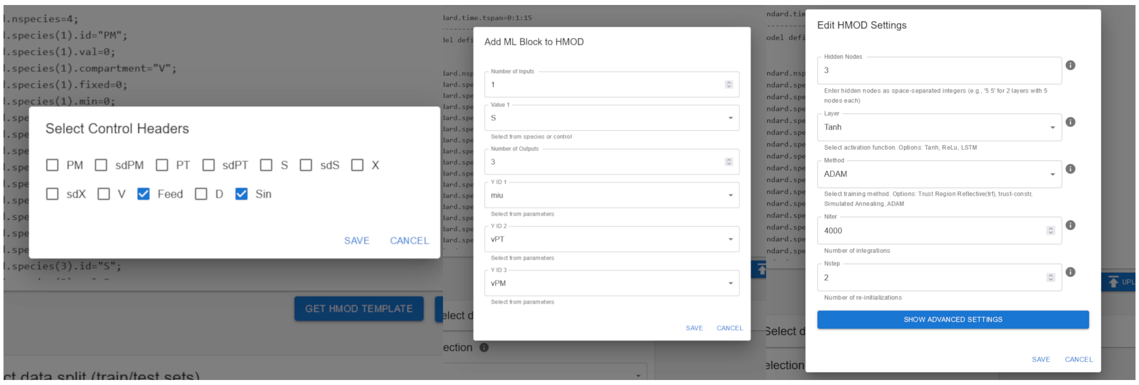


Figure 5.1: Example of a species time-series representation for a specific experimental dataset. The panel appears when the users upload their data `.csv` file and click the "View Batches" button. The data represents the first "batch of data" overview used for the *Park and Ramirez* model training. The synthetic data came from MATLAB then a "normal" noise was applied on each point depending on its standard deviation.

Uploading and Editing Model Information When a user uploads an `.hmod` file, the system will check whether all required sections related to the machine learning model are present. If any essential information is missing, a modal dialog will automatically prompt the user to input the missing details, ensuring that all parameters are correctly defined. This dynamic form validation helps users maintain the integrity of the model's configuration. Once the model is complete, users can revisit and edit the details via a dedicated "Edit HMOD Settings" button next to the upload feature. This button lets users make changes or review the model configuration at any process stage.

Ensuring Configuration Completeness The website includes functionality to ensure that all sections of the `.hmod` files are correctly configured. For instance, it checks if the control section is present and prompts users to fill in details if they are missing. Additionally, it guides users through configuring machine learning model layers, hidden nodes, and other parameters essential for accurate model training. Figure 5.2 demonstrates these checks.


```
.nspecies=4;
.species(1).id="PM";
.species(1).val=0;
.species(1).compartment="V";
.species(1).fixed=0;
.species(1).min=0;
.species(3).id="S";
```

Select Control Headers

PM [ ] sdPM [ ] PT [ ] sdPT [ ] S [ ] sdS [ ] X [ ] sdX [ ] V [ ] [✓ Feed] D [ ] [✓ Sin]

Add ML Block to HMOD

- Number of Inputs: 1
- Value 1: S — Select from species or control
- Number of Outputs: 3
- Y ID 1: miu — Select from parameters
- Y ID 2: vPT — Select from parameters
- Y ID 3: vPM — Select from parameters

Edit HMOD Settings

- Hidden Nodes: 3 — Enter hidden nodes as space-separated integers (e.g., '5 5' for 2 layers with 5 nodes each)
- Layer: Tanh — Select activation function. Options: Tanh, ReLu, LSTM
- Method: ADAM — Select training method. Options: Trust Region Reflective(trf), trust-constr, Simulated Annealing, ADAM
- Niter: 4000 — Number of integrations
- Nstep: 2 — Number of re-initializations

Figure 5.2: Pop-up windows required for the user to fill in so that the `.hmod` file contains all the necessary information for training the model. From left to right: the first image allows the selection of control variables taken from the `.csv` file header; the second image enables the user to specify the desired number of inputs and outputs and select what those inputs and outputs are; the third image displays the ML settings, where the user can adjust parameters such as the number of hidden nodes, activation functions, training methods, etc. after all necessary information is available in the `.hmod` file.

Batch Selection for Training and Testing Users must select the type of batch selection for training and testing their model:

- **Manual (Mode 1):** Users manually select specific batches for training and testing. This mode allows precise control over which data is used for training and which is reserved for testing.
- **Random (Mode 2):** Batches are randomly selected, with a typical split of 2/3 for training and 1/3 for testing. This model provides a balanced and automated approach to batch selection.

Uploading and Processing Once all necessary files are uploaded and configurations are set, users can initiate the model training process by clicking the upload button. The system validates the inputs, uploads the data, and starts the training session. Users can monitor the progress and view detailed logs of the training process.

This comprehensive approach ensures that users can efficiently manage their model configurations and datasets, facilitating a streamlined and error-free training process.

5.1.4 Simulation Page

The "Simulation" Page closely mirrors the structure and functionality of the "New Project" Page, with a primary focus on allowing users to simulate models using previously trained HMOD files. The critical difference is that, in addition to uploading new HMOD files, users can select and use already trained HMOD files from past runs. This enhances the system's flexibility and usability by enabling users to revisit trained models and perform simulations of results with different data sets without the need to retrain the model.

5.1.5 Results Page

The "Results" page provides an overview of the most recently trained or simulated model. This page is particularly useful for users who prefer a streamlined and less cluttered interface to review their model's performance. By offering a concise presentation of key metrics and outputs, the Results page enables users to quickly assess the outcomes of their training or simulation without the distractions of additional interface elements.

5.1.6 Historical Page

The "Historical" page provides users with a comprehensive list of all previous training sessions performed on the website. Each entry in this list includes essential details such as the training title, the names of the `.hmod` and `.csv` files used, and the batch selection mode. Clicking on an entry opens a modal that displays detailed information about the training, including machine learning options selected by the user, training graphs, metrics, and the resulting array of weights and biases of the network.

Here is a detailed explanation of the process:

Overview of Past Training Sessions The "Historical" page is designed to provide users with easy access to information about their previous training sessions. This includes vital details such as the training title, the names of the files used, and the batch selection mode. This overview helps users quickly identify and select specific training sessions for further analysis.

Detailed Information Modal When a user clicks on a training session entry, a modal window displays comprehensive details about the selected training session. This modal includes:

- **Title:** Title of the training session.
- **Plots:** Interactive plots show the performance of the model after training.
- **Training Metrics:** Graphs and metrics illustrating the performance of the trained model.
- **Weights and Biases:** The resulting array of weights and biases of the network is presented in a tabular format.
- **Hmod File:** The name of the `.hmod` file used in the session.
- **CSV File:** The name of the `.csv` file used in the session.
- **NewHmod File:** The name of the `.newhmod` file generated after training.
- **Mode:** The batch selection mode (manual or random).
- **Machine Learning Options:** Detailed parameters and options used during the training.

Interactive Visualization The modal includes interactive elements that allow users to view and analyze the training plots. Users can click on individual plots to open a larger view, facilitating detailed examination of the training data and model performance. Two types of plots will be available for analysis:

- **Metabolite Plots:** These plots are line graphs that compare the observed data with the model's predicted data. This visualization helps in understanding how well the model is performing in predicting the behavior of metabolites over the training data set. The x-axis represents time, while the y-axis shows the concentration levels of the metabolites.
- **Predicted vs Observed Plots:** These plots illustrate the relationship between the predicted values generated by the model and the actual values from the data set. Points on these scatter plots ideally lie on or near the line of perfect prediction (where predicted values equal actual values). Deviations from this line indicate the model's prediction errors, providing insights into the model's accuracy and potential areas for improvement.

5.1.7 Help Page

The Help Page provides users with detailed information on how to use the website, including step-by-step guides on how to use the different features and functionalities. This page is a comprehensive resource for users, offering detailed explanations and instructions on navigating the website, uploading files, training models, and interpreting results. The Help Page is designed to provide users with the necessary information to make the most of the website's capabilities and features, ensuring a seamless and user-friendly experience.

5.2 Usage Examples

This section describes the results obtained from the case studies selected to test/validate the platform and the key capabilities. To illustrate the main features of the application, three different dynamic models were used: the *Chassagnole* model described in [63], the *Park and Ramirez* model described in [64], and a new *Basic* model. Each model was chosen for specific reasons that will be addressed below, allowing us to evaluate the performance of the HYBpy tool.

For consistency, the models were subjected to the same number of integration steps, utilized the *Tanh* activation function, and shared the same stopping criteria: ($f_{\text{tol}} = 1\times10^{-10}$, $x_{\text{tol}} = 1 \times 10^{-10}$ and $g_{\text{tol}} = 1 \times 10^{-10}$).

These stopping criteria are standard in optimization algorithms and are used to control the precision of the solution:

- **ftol**: The relative tolerance for the change in the cost function value. Optimization will stop when the difference between consecutive function values is smaller than this value. A small value like 1×10^{-10} ensures high accuracy in the cost function minimization.

- **xtol**: The relative tolerance for the change in the solution vector. Optimization will stop if the relative change in the solution vector between iterations is less than 1×10^{-10}, indicating that the solution has converged.

- **gtol**: The relative tolerance for the gradient. Optimization will stop when the norm of the gradient is less than 1×10^{-10}, ensuring that the optimization has reached a point where the gradient is close to zero, which indicates a local minimum.

The performance of the models was evaluated using the Mean Squared Error (MSE) as the key performance indicator. The results of these experiments are presented in the following sections.

5.2.1 Selection of Models

The bioreactor models were chosen intentionally to test HYBpy's adaptability and robustness in dealing with varying levels of model complexity and to illustrate its application in various bioprocess settings.

1. ***Chassagnole* Model**: The Chassagnole model describes the threonine synthesis pathway in E. coli. This model simulates the time course of 11 species in a single compartment, making it very complex. This complexity makes it an ideal candidate

for evaluating HYBpy's ability to handle sophisticated mechanistic models and effectively integrate machine learning components.

2. ***Park and Ramirez* Model**: The *Park and Ramirez* model focuses on a bioreactor system with dynamic behavior suitable for testing optimization algorithms. It involves four species and incorporates control mechanisms, representing moderate complexity. This model allows HYBpy's performance to be gauged in handling models of intermediate complexity and testing different training algorithms and configurations.

3. ***Basic* Model**: The *Basic* model is a simplified bioreactor model developed from scratch. It represents a simple case of a simple fed-batch model that describes only the dynamics of viable cell density (VCD) and substrate concentration (Sub). This model was included to demonstrate how HYBpy can be used to build a model from the ground up, meaning this models was created from an empty hmod file, highlighting the tool's user-friendly features and flexibility in handling basic modeling scenarios. It serves as a baseline for comparing HYBpy's performance with more complex models. These are the ODEs equations for the *Basic* Model:

   a) Biomass growth:
   $$\frac{dVCD}{dt} = (\frac{Vmax \cdot Sub}{(Ks + Sub)}) \cdot VCD \tag{5.1}$$

   b) Substrate consumption:
   $$\frac{dSub}{dt} = -(\alpha \cdot \mu + \beta \cdot Sub) \cdot VCD + feed\_rate \tag{5.2}$$

   Where, μ is the specific growth rate, given by Monod equation; α and β are empirical parameters.

Table 5.1: Overview of the three models used.

Model	Number of Species	Number of Reactions	Number of Parameters	Reference
Chassagnole	11	7	40	[63]
Park and Ramirez	4	4	4	[64]
Basic model	2	3	6	Our Model

5.2.2 Steps for Training

The steps to achieve the results will be presented below. We will use the *Park and Ramirez* Model as an example, but the steps are the same for all models. Figure 5.3 illustrates the complete sequence for initializing a model training session.

The steps are as follows:

1. Navigate to the "New Project" Page: Begin by selecting the "New Project" page from the menu on the left side of the interface to start the training process

2. Set a Training Title: Give a title to the training session you are about to create.
3. Upload CSV File: After entering the title, the button to upload a `.csv` file will be enabled. Click the button "Upload CSV" and by precessing select the `.csv` file you wish to utilize. Once the file is uploaded, if necessary, you can view the experimental data divided by batches in a graphical format by clicking on the "View Batches" button, as shown in Figure 5.1.
4. Upload and Configure HMOD File: When the `.csv` is uploaded correctly, proceed to the upload of the `hmod` file. If the `.hmod` does not have control information, a pop-up window will appear, prompting you to provide this information to proceed with the creation of the machine learning (ML) block (see Figure 5.2). Similarly, if the ML block is missing, another pop-up will prompt you to input the essential ML configuration details (see Figure 5.2).
5. Adjust Training Settings: When the `.hmod` has all the necessary information, the user can modify the training parameters as desired. This includes adjusting settings such as the number of hidden nodes, activation function, training method, and other ML-related configurations (refer to Figure 5.2).
6. Select Batch selection and Start Training After all that, the user must select which batch selection to use for training. The training process will be initiated by pressing the "Start Training" button.

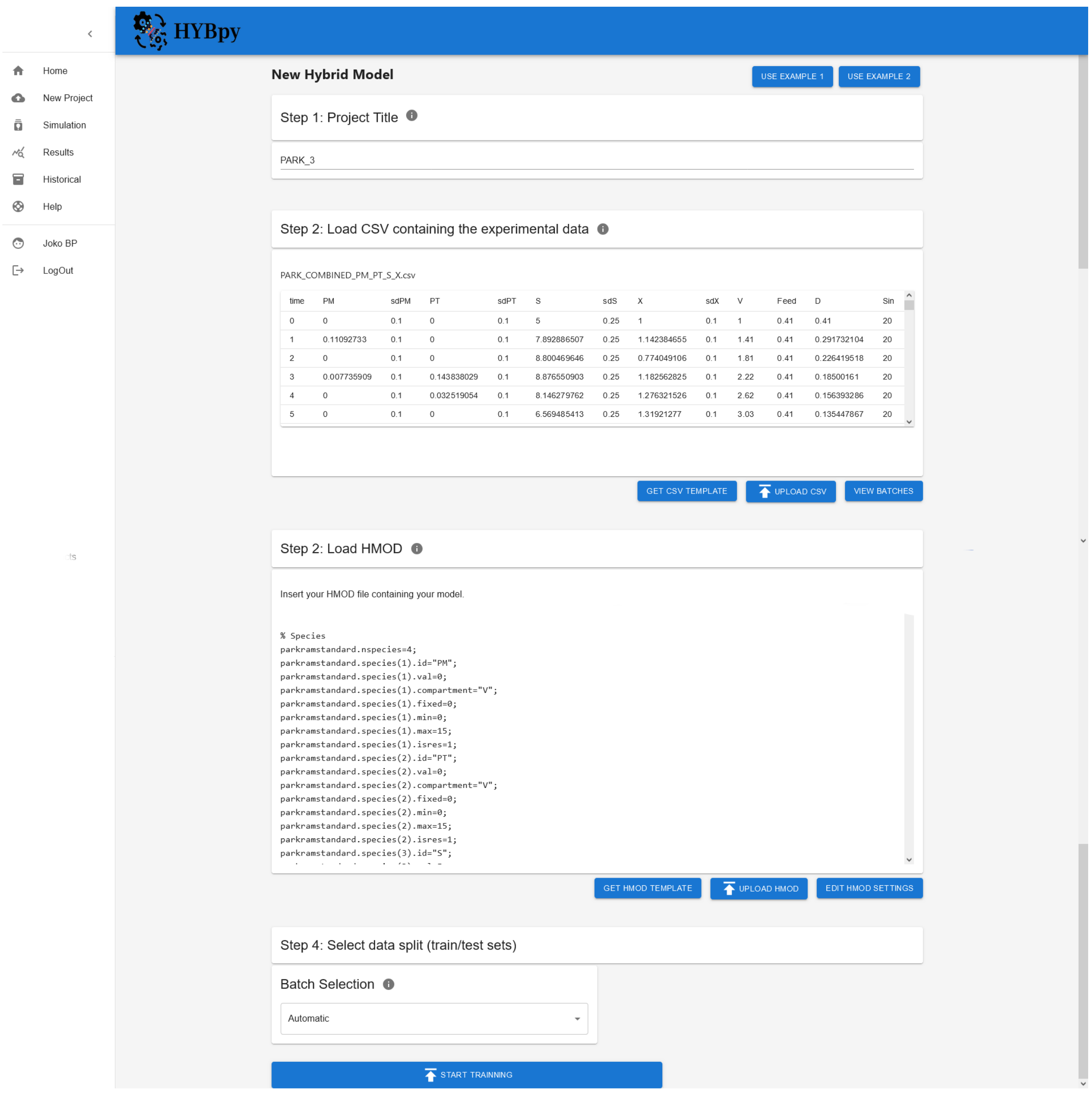
5.2.3 Models Configurations

Each model was trained using specific machine learning algorithms and neural network architectures tailored to its complexity and characteristics.

5.2.3.1 *Chassagnole* Model

The *Chassagnole* model was trained using the Trust Region Reflective (TRF) algorithm. TRF is well-suited for nonlinear least squares problems and effectively handles large-scale optimization tasks. The neural network architecture consisted of two hidden layers with five neurons each, using the *Tahn* activation function.

The TRF algorithm was chosen for its simplicity and suitability for linear models, making it an efficient and straightforward optimization method for the *Chassagnole* model. The neural network architecture with two hidden layers, each containing five neurons, was deliberately selected to showcase HYBpy's ability to build deep learning networks. The low Mean Squared Error (MSE) values and high R^2 scores on both training and testing datasets indicate that the hybrid model accurately captures the dynamics of the *Chassagnole* system. This experiment shows that HYBpy can efficiently train linear models using simple optimization algorithms like TRF while supporting deep learning network



New Hybrid Model

Step 1: Project Title

PARK_3

Step 2: Load CSV containing the experimental data

PARK_COMBINED_PM_PT_S_X.csv

time	PM	sdPM	PT	sdPT	S	sdS	X	sdX	V	Feed	D	Sin
0	0	0.1	0	0.1	5	0.25	1	0.1	1	0.41	0.41	20
1	0.11092733	0.1	0	0.1	7.892886507	0.25	1.142384655	0.1	1.41	0.41	0.291732104	20
2	0	0.1	0	0.1	8.800469646	0.25	0.774049106	0.1	1.81	0.41	0.226419518	20
3	0.007735909	0.1	0.143838029	0.1	8.876550903	0.25	1.182562825	0.1	2.22	0.41	0.18500161	20
4	0	0.1	0.032519054	0.1	8.146279762	0.25	1.276321526	0.1	2.62	0.41	0.156393286	20
5	0	0.1	0	0.1	6.569485413	0.25	1.31921277	0.1	3.03	0.41	0.135447867	20

Step 2: Load HMOD

Insert your HMOD file containing your model.

```
% Species
parkramstandard.nspecies=4;
parkramstandard.species(1).id="PM";
parkramstandard.species(1).val=0;
parkramstandard.species(1).compartment="V";
parkramstandard.species(1).fixed=0;
parkramstandard.species(1).min=0;
parkramstandard.species(1).max=15;
parkramstandard.species(1).isres=1;
parkramstandard.species(2).id="PT";
parkramstandard.species(2).val=0;
parkramstandard.species(2).compartment="V";
parkramstandard.species(2).fixed=0;
parkramstandard.species(2).min=0;
parkramstandard.species(2).max=15;
parkramstandard.species(2).isres=1;
parkramstandard.species(3).id="S";
```

Step 4: Select data split (train/test sets)

Batch Selection

Automatic

Figure 5.3: Screenshot illustrating the step-by-step process for training the *Park and Ramirez* model. The sequence includes navigating to the "New Project" page, setting a training title, uploading and configuring a CSV and `.hmod` file, adjusting training settings, selecting batch selection, and initiating training. Each step is depicted to guide the user through the setup to the start of model training.

development. The performance of the trained model was evaluated using the Mean Squared Error (MSE) and the coefficient of determination (R^2) on both training and testing datasets. The results are as follows:

Table 5.2: Performance metrics for the *Chassagnole* model. Training results are based on three training batches and two validation batches. The hybrid model was trained using the Trust Region Reflective (TRF) method with the hyperbolic tangent (*Tanh*) activation function, incorporating cross-validation (CV) and indirect sensitivities. The neural network architecture consisted of two hidden layers with five neurons each. A tau value of 0.25 was used, and the training was conducted with 100 as the max number of objective function evaluations.

Training MSE	Testing MSE	Training R^2	Testing R^2
1.3227×10^{-5}	1.5253×10^{-5}	0.9952	0.9975

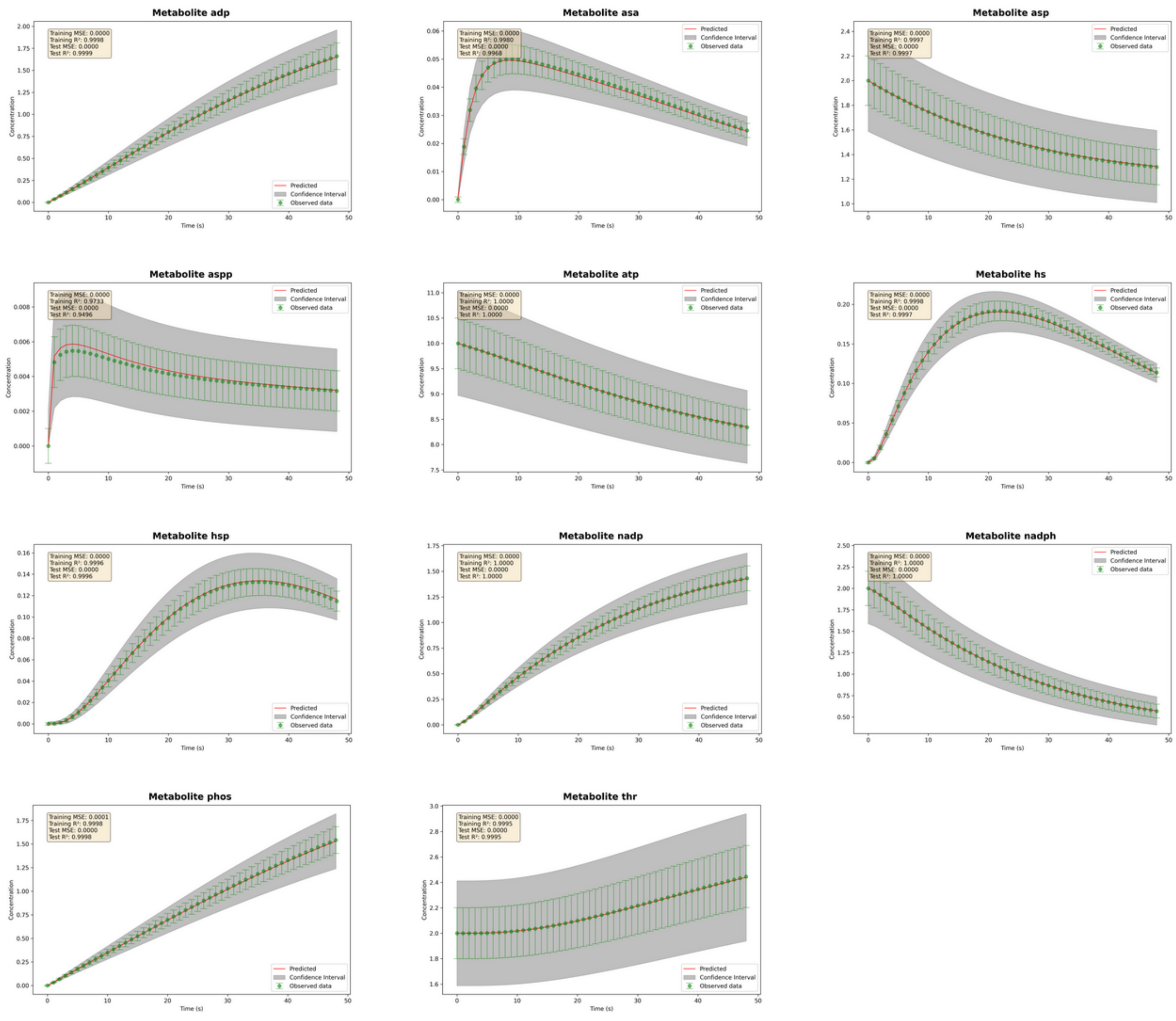


Figure 5.4: Comparison of the *Chassagnole* model training results with experimental data, as described in the text. The green dots represent observations and their respective ± standard deviation. The red line represents the model prediction. The values were obtained using the same training steps as described in 5.2.2

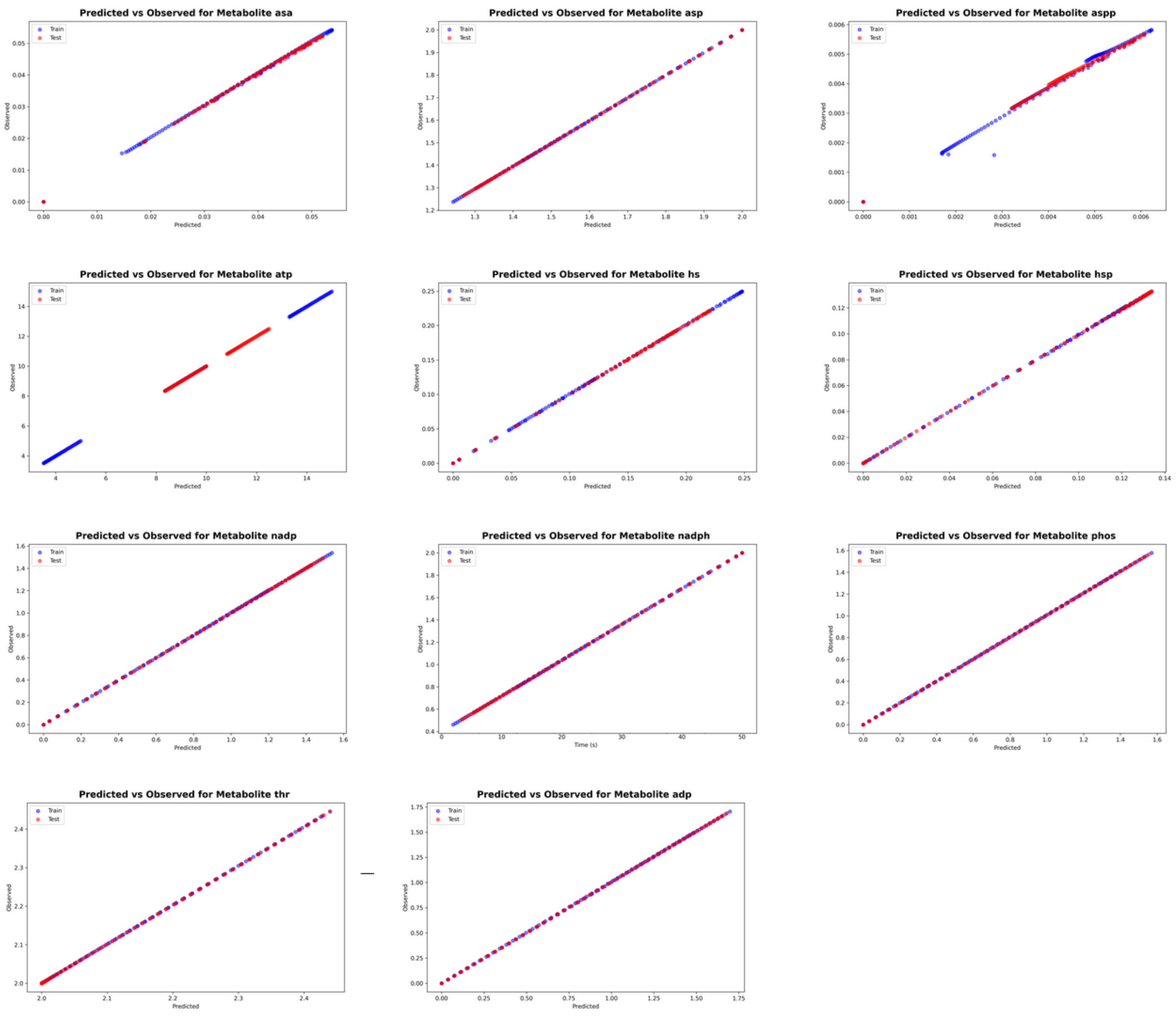


Figure 5.5: Plots for the *Chassagnole* predicted values versus observed values. The blue dots represent the prediction for the data marked for the train, and the red for the test.

5.2.3.2 *Park and Ramirez* Model

The *Park and Ramirez* model was trained using ADAM optimization algorithm. The neural network used to train has a single hidden node layer with three neurons, using the *Tahn* activation function.

The ADAM optimization algorithm was chosen for its efficiency and robustness in training neural networks, especially for models with advantageous gradient descent methods. It combines the benefits of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), allowing for faster convergence and improved performance on noisy data. A neural network architecture with a single hidden layer of three neurons was deliberately chosen to maintain simplicity and prevent overfitting. Given the moderate complexity of the *Park and Ramirez* model, a simple network was sufficient to capture the underlying patterns without introducing unnecessary complexity. The low MSE values and high R^2 scores indicate that the hybrid model accurately captured the dynamics of the *Park and Ramirez* system. The testing R^2 value of 0.9803 suggests that the model explains a significant proportion of the variance in the testing data, demonstrating strong generalization capabilities. The slightly lower training R^2 value of 0.8998 may be due to the stochastic nature of the ADAM optimizer. This experiment highlights HYBpy's ability to effectively train hybrid models using different optimization algorithms and neural network configurations. The successful implementation of the ADAM optimizer and a simple neural network architecture demonstrates HYBpy's flexibility and suitability for a variety of bioprocess modeling scenarios. The performance of the trained model was evaluated using the Mean Squared Error (MSE) and the coefficient of determination (R^2) on both training and testing datasets. The results are as follows:

Table 5.3: Performance metrics for the *Park and Ramirez* model. The hybrid model was trained using the ADAM optimization method with the hyperbolic tangent (*Tanh*) activation function incorporating cross-validation (CV) and indirect sensitivities. The neural network architecture consisted of a single hidden layer with three neurons. A tau value of 0.3 was used, and a total of 4000 iterations. The training was conducted on ten batches, with five batches used for validation.

Training MSE	Testing MSE	Training R^2	Testing R^2
0.0331	0.0305	0.8998	0.9803

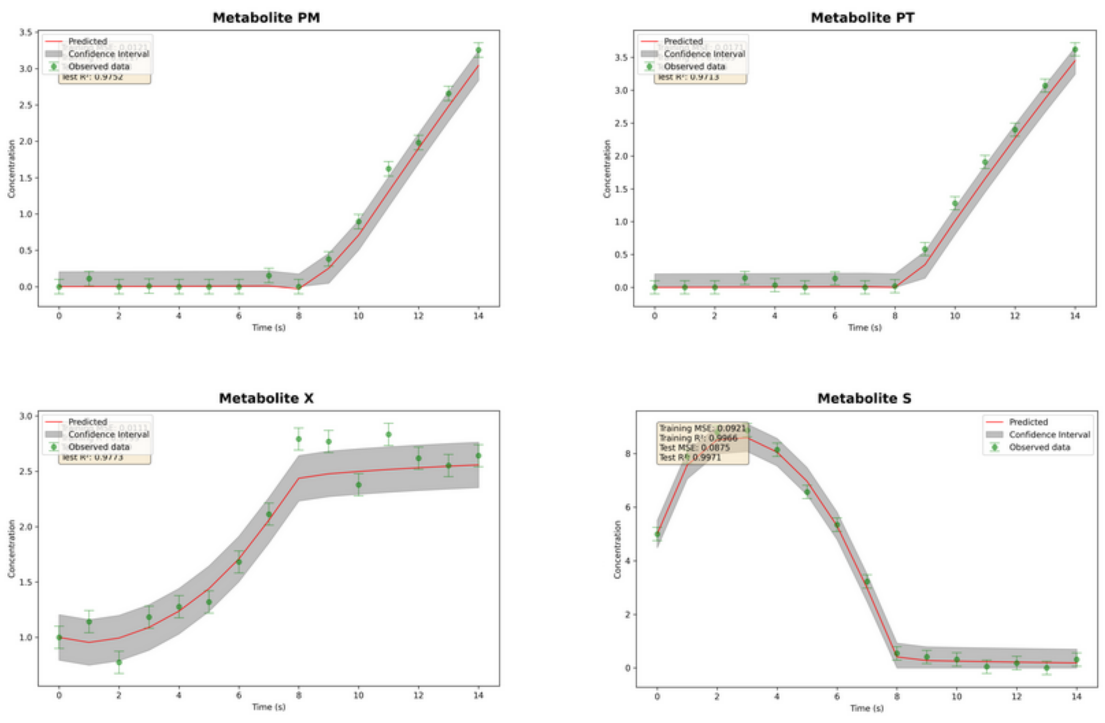


Figure 5.6: Comparison of the *Park and Ramirez* model training results with experimental data, as described in the text. The green dots represent observations and their respective ± standard deviation. The red line represents the model prediction. The values were obtained using the same training steps as described in 5.2.2

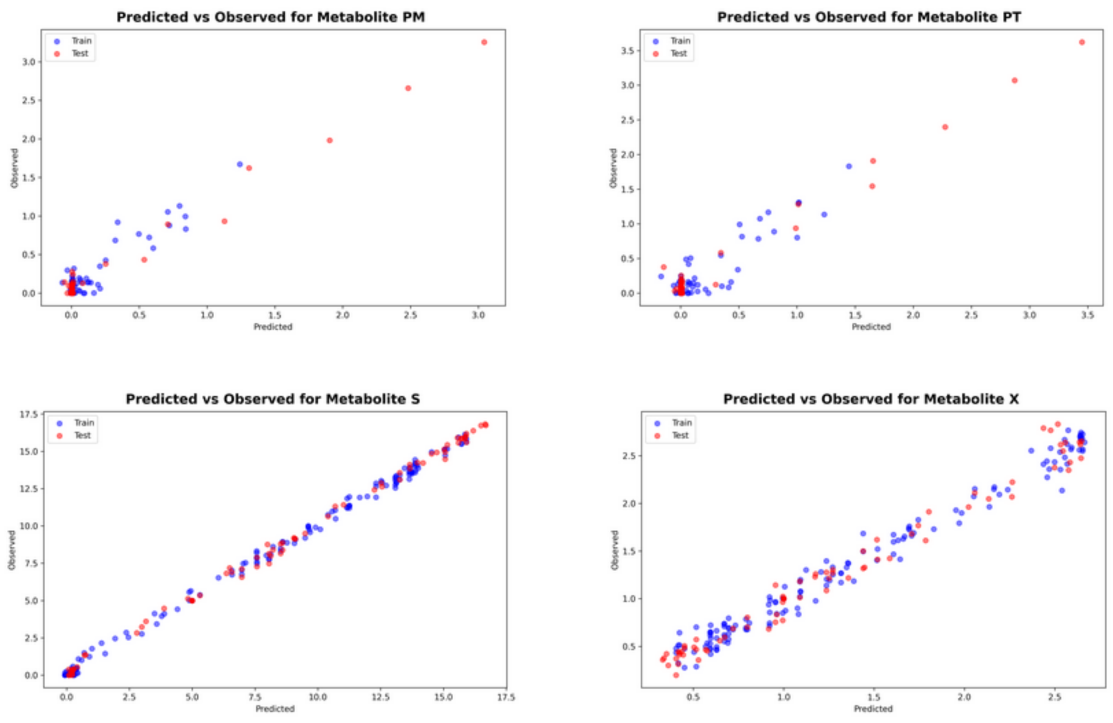


Figure 5.7: Plots for the *Park and Ramirez* predicted values versus observed values. The blue dots represent the prediction for the data marked for the train, and the red for the test.

5.2.3.3 *Basic* Model

The *Basic* model was trained using the Trust Region Reflective (TRF) algorithm, similar to the Chassagnole model. The neural network architecture consisted of a single hidden layer with five neurons, utilizing the *Tanh* activation function. This configuration was chosen to balance simplicity and the ability to capture the nonlinear dynamics present in the *Basic* model.

The choice of the TRF algorithm was motivated by its simplicity and efficiency for models with linear or mildly nonlinear behavior. Since the *Basic* model represents a simplified bioreactor system with straightforward dynamics, TRF was an appropriate choice that ensured reliable convergence without unnecessary computational complexity.

The neural network architecture with a single hidden layer of five neurons was selected to provide sufficient capacity to model the underlying dynamics without overfitting. Using more neurons or additional layers might not significantly improve performance for such a simple model and could introduce unnecessary complexity.

The performance of the trained model was evaluated using the Mean Squared Error (MSE) and the coefficient of determination (R^2) on both the training and testing datasets. The results are presented in Table 5.4.

Table 5.4: Performance metrics for the *Basic* model. Training results are based on ten training batches and five validation batches. The hybrid model was trained using the Trust Region Reflective (TRF) method with the hyperbolic tangent (*Tanh*) activation function, incorporating cross-validation (CV) and indirect sensitivities. The neural network architecture consisted of one hidden layer with five neurons. A tau value of 0.3 was used, and the training was conducted with 4000 as the max number of objective function evaluations.

Training MSE	Testing MSE	Training R^2	Testing R^2
0.3733	0.3385	0.9547	0.9617

The low MSE values on both training and testing datasets indicate that the model accurately captured the dynamics of the *Basic* bioreactor system. The high R^2 values, exceeding 0.95, demonstrate that the model explains a large proportion of the variance in the data, confirming its predictive power and generalization capability.

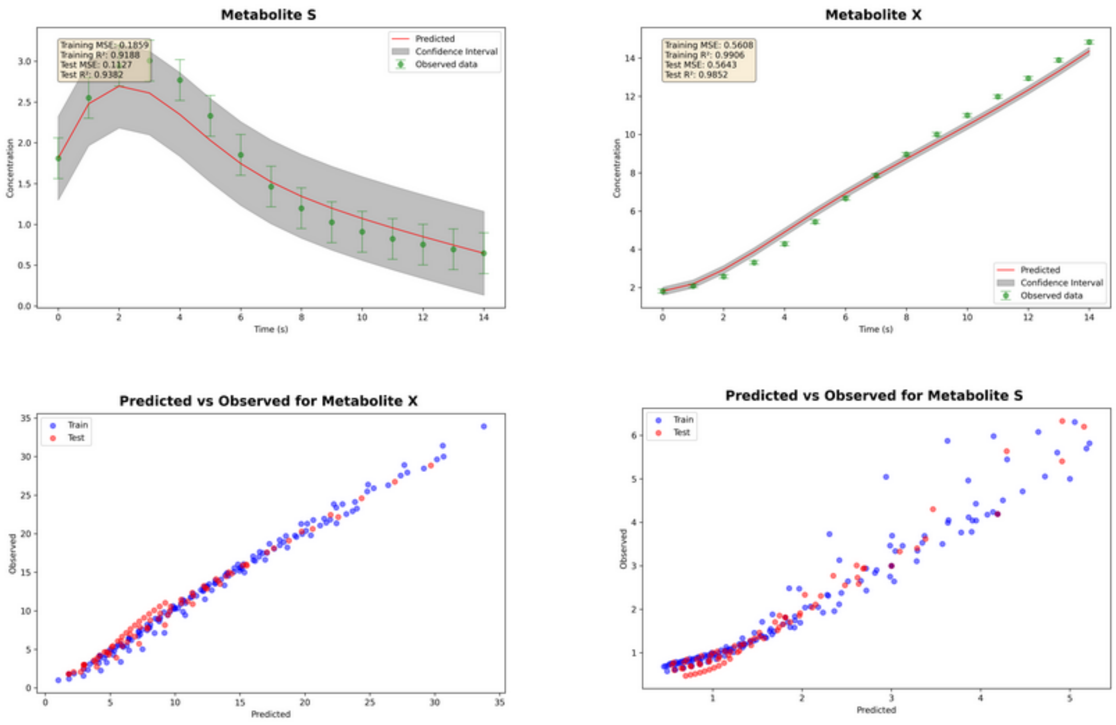


Figure 5.8: First two graph are the comparison of the *Basic* model training results with experimental data, as described in the text. The green dots represent observations and their respective ± standard deviation. The red line represents the model prediction. The values were obtained using the same training steps as described in 5.2.2. The last two plots are for the *Basic* predicted values versus observed values. The blue dots represent the prediction for the data marked for the train, and the red for the test.

5.2.4 Comparison and Analysis

Overall Performance: The experiments conducted on the three bioreactor models—the *Chassagnole* model, the *Park and Ramirez* model, and the *Basic* model—demonstrate the versatility and effectiveness of HYBpy in handling models of varying complexity. The key performance metrics for each model are summarized in Table 5.5.

Table 5.5: Performance Metrics for the Bioreactor Models.

Model	Training MSE	Testing MSE	Training R^2	Testing R^2
Chassagnole	1.3227×10^{-5}	1.5253×10^{-5}	0.9952	0.9975
Park and Ramirez	0.0331	0.0305	0.8998	0.9803
Basic	0.3733	0.3385	0.9547	0.9617

Chassagnole Model: The hybrid model achieved very low MSE values on training and testing datasets, with R^2 scores of 0.99. This indicates excellent predictive accuracy and generalization capability, effectively capturing the complex dynamics of the threonine synthesis pathway.

Park and Ramirez Model: The model showed low MSE values and high R^2 scores, particularly on the testing dataset, where the R^2 value reached 0.9803. This demonstrates that the model generalizes well to unseen data despite having a slightly lower training R^2.

Basic Model: The Basic model also displayed high R^2 scores and low MSE values on the training and testing datasets. With R^2 values of 0.9547 and 0.9617, respectively, the training and testing MSEs were 0.3733 and 0.3385. These findings show that the model successfully extended to new data and faithfully represented the straightforward dynamics of the fundamental bioreactor system. Only one hidden layer with five neurons and the TRF algorithm was needed to adequately simulate the system.

In Sum, the HYBpy case studies showcase its effectiveness as a tool for developing and training hybrid models in the bioprocess domain. The key features are:

- Versatility: HYBpy successfully handled varying complex models. This versatility makes it suitable for a wide range of applications.

- Ease of Use: With its web-based interface and step-by-step pipeline, HYBpy is accessible to users without extensive programming knowledge. This lowers the barrier to entry for researchers and practitioners in the field.

- Performance: The models trained using HYBpy demonstrated solid performance, with high R^2 values and low MSEs on both training and testing datasets. However, it was observed that the training process using HYBpy required more time than the MATLAB implementation. In Chapter 6: Conclusion and Future Work 6, we will explore potential strategies to enhance the training efficiency of HYBpy.

- Customization: Users can tailor the neural network architecture and select appropriate optimization algorithms based on the specific needs of their models, providing a high degree of customization.

Why Should Users Use HYBpy? HYBpy offers several advantages, making it an attractive tool for researchers and process engineers working on hybrid modeling. By providing an accessible, flexible, and powerful platform for hybrid modeling, HYBpy addresses a significant gap in the current landscape of modeling tools. It empowers users to develop accurate and adaptable models more efficiently, accelerating research and innovation in the bioprocess field.

6 Conclusion & Future Work

6.1 Main Conclusions

In recent years, the development and adoption of hybrid models have increased significantly, driven by the need to improve the accuracy and adaptability of models in various fields, namely bioprocesses development and biological systems analysis. This work proposes a web-based hybrid modeling tool HYBpy (www.hybpy.org), that aims to facilitate the development and training of hybrid models. This framework provides a comprehensive platform for hybrid model construction, simulation, and analysis, combining state-of-the-art techniques in both fields.

The main accomplishment of this work is creating a user-friendly and open-source tool capable of building and simulating hybrid models. The HYBpy tool enables researchers without extensive programming knowledge to develop hybrid models for bioprocesses and biological systems applications. The tool's web-based interface allows users to create and train hybrid models using a step-by-step pipeline, making it accessible to a broader audience in a MATLAB free fashion. The tool also supports a variety of simulation options and has significantly enhanced the user experience by incorporating features such as easy file management, model previewing, and data visualization.

HYBpy advances the state of hybrid modeling by integrating deep learning and mechanistic modeling techniques into a seamless framework. It also supports the FAIR (Findable, Accessible, Interoperable, and Reusable) principles by facilitating standardized experimental data input and model management. As more researchers adopt HYBpy, its capabilities will continue to expand, enabling the development of accurate and adaptable hybrid models in the bioprocess field.

6.2 Future Directions and Limitations

This section highlights potential enhancements for HYBpy, including new functionalities and enhancements that could be implemented in future versions, based on the current achievements.

6.2.1 Faster Model Training

Currently, the training process for hybrid models can be time-consuming, especially for large models/datasets. While Python offers flexibility and accessibility, CPU-bound operations can introduce bottlenecks in performance, especially when handling the computationally intensive tasks associated with training deep learning models.

To address this, two primary strategies can be implemented to improve the training speed of hybrid models:

- **GPU Acceleration:** By leveraging GPUs for training, HYBpy can benefit from the parallel processing capabilities of these devices, significantly reducing the time required for model training. Libraries such as PyTorch already support GPU acceleration, and migrating the current calculations to run on GPUs would allow HYBpy to handle larger datasets and more complex models more efficiently.

- **Optimizing Tensor Sizes:** The existing HYBpy implementation uses `torch.float64` tensors offer high precision but have higher computational complexity and memory usage. In numerous cases, decreasing the precision to `torch.float32` or smaller types could result in substantial performance improvements without compromising accuracy for most tasks. Optimizing tensor sizes can reduce memory usage and processing time, enhancing the speed and efficiency of training hybrid models.

6.2.2 Expand Model Libraries

A crucial step in advancing the HYBpy framework and ensuring its continued relevance in the field of hybrid modeling is the expansion of its model library. The current iteration of HYBpy offers a limited selection of model types, primarily focused on fundamental deep-learning architectures and mechanistic modeling approaches. To enhance its versatility and applicability to a wider range of research endeavors, future development should prioritize integrating more sophisticated models, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformer models. Incorporating these advanced architectures would facilitate the development of more intricate and precise hybrid models capable of addressing complex tasks. HYBpy's modular design is inherently conducive to the seamless integration of new model types, ensuring that such updates can be implemented without necessitating extensive code refactoring. Furthermore, including state-of-the-art techniques such as reinforcement learning and transfer learning would significantly augment HYBpy's capabilities. These techniques offer advanced training and optimization strategies, enabling the development of more efficient and adaptive models. By supporting these emerging methodologies, HYBpy would remain at the forefront of hybrid modeling, empowering researchers to address increasingly sophisticated challenges within this evolving domain.

6.2.3 Integrate SBML2HYB

In order to transform HYBpy into a full, all-in-one solution for researchers and practitioners, the functionality of SBML2HYB must be integrated directly into the HYBpy tool. Currently, users must use SBML2HYB to convert SBML files into `.hmod` files without the machine learning (ML) block. Then, upload the `.hmod` files to the HYBpy web interface to add the ML block and start model training.

By integrating SBML2HYB's capabilities into HYBpy, users will be able to submit SBML files directly to the HYBpy interface and generate the associated `.hmod` files and seamlessly incorporate the ML block inside the platform. This update will significantly streamline the workflow, reducing the complexity of the hybrid model development process. It will also make the tool more accessible to users unfamiliar with the intricacies of model conversion, fostering greater acceptance of HYBpy within the hybrid modeling community.

Users will find the modeling process with HYBpy to be intuitive and efficient, fostering greater collaboration and innovation in developing hybrid models. Future iterations of HYBpy will continue to prioritize user-friendliness, providing a more cohesive and enjoyable experience.

6.3 Scientific Contribution

Our research, utilizing the HYBpy framework, has been selected for a poster presentation at the 14th European Symposium on Biochemical Engineering Sciences (ESBES 2024) [86]. This presentation will demonstrate HYBpy's advanced capabilities in hybrid modeling and its significant implications for the field. Being featured at such a prestigious international event underscores the groundbreaking nature of our work and its potential to influence future practices in biochemical engineering.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: `https://github.com/joaomlourenco/novathesis/raw/main/template.pdf` (cit. on p. i).

[2] H. Narayanan et al. *Hybrid modeling for biopharmaceutical processes: advantages, opportunities, and implementation*. 2023. DOI: `10.3389/fceng.2023.1157889` (cit. on pp. 1, 2, 11, 14).

[3] R. ltd and Markets. *Global Biopharmaceuticals Market Report and forecast 2023-2031*. URL: `https://www.researchandmarkets.com/reports/5805734/global-biopharmaceuticals-market-report-forecast` (cit. on p. 1).

[4] G. Walsh. "Biopharmaceutical benchmarks 2018". In: *Nature Biotechnology* 36 (12 2018-12), pp. 1136–1145. ISSN: 15461696. DOI: `10.1038/nbt.4305` (cit. on p. 1).

[5] A. L. Grilo and A. Mantalaris. "The Increasingly Human and Profitable Monoclonal Antibody Market". In: *Trends in Biotechnology* 37 (1 2019-01), pp. 9–16. ISSN: 0167-7799. DOI: `10.1016/J.TIBTECH.2018.05.014` (cit. on p. 1).

[6] M. E. Lalonde and Y. Durocher. "Therapeutic glycoprotein production in mammalian cells". In: *Journal of Biotechnology* 251 (2017-06), pp. 128–140. ISSN: 0168-1656. DOI: `10.1016/J.JBIOTEC.2017.04.028` (cit. on p. 1).

[7] G. Walsh and E. Walsh. "Biopharmaceutical benchmarks 2022". In: (). DOI: `10.1038/s41587-022-01582-x`. URL: `https://doi.org/10.1038/s41587-022-01582-x` (cit. on p. 1).

[8] H. Narayanan et al. *Machine Learning for Biologics: Opportunities for Protein Engineering, Developability, and Formulation*. 2021-03. DOI: `10.1016/j.tips.2020.12.004` (cit. on p. 1).

[9] A. G. Cardillo et al. *Towards in silico Process Modeling for Vaccines*. 2021-11. DOI: `10.1016/j.tibtech.2021.02.004` (cit. on p. 1).

[10] H. Narayanan and C. Love. "Process modeling in the CMC of vaccines: are we doing it right?" In: *Vaccine Insights* 01 (05 2022-11), pp. 299–314. ISSN: 27525422. DOI: `10.18609/vac.2022.042` (cit. on p. 1).

[11] A. S. Rathore and S. Nikita. "Digitalization: The Route to Biopharma 4.0." In: 36.11 (2023), 25–29. URL: `https://cdn.sanity.io/files/0vv8moc6/biopharn/c2b39dddé6b7352539bd2a0c33d9c6ed172d2397.pdf/BP1123_ezine%20(Watermark)%20LINKED.pdf` (cit. on p. 1).

[12] S. C. James, R. L. Legge, and H. Budman. "ON-LINE ESTIMATION IN BIOREACTORS: A REVIEW". In: *Reviews in Chemical Engineering* 16.4 (2000), pp. 311–340. DOI: `doi:10.1515/REVCE.2000.16.4.311`. URL: `https://doi.org/10.1515/REVCE.2000.16.4.311` (cit. on pp. 2, 3).

[13] M. K. Alavijeh et al. "Digitally enabled approaches for the scale up of mammalian cell bioreactors". In: *Digital Chemical Engineering* 4 (2022-09), p. 100040. ISSN: 2772-5081. DOI: `10.1016/J.DCHE.2022.100040` (cit. on p. 3).

[14] M. von Stosch et al. "Hybrid modeling for quality by design and PAT-benefits and challenges of applications in biopharmaceutical industry". In: *Biotechnology Journal* 9 (6 2014), pp. 719–726. ISSN: 18607314. DOI: `10.1002/biot.201300385` (cit. on pp. 3, 8).

[15] M. V. Stosch, J.-M. Hamelink, and R. Oliveira. "Hybrid modeling as a QbD/PAT tool in process development: an industrial E. coli case study". In: *Bioprocess and Biosystems Engineering* 39 (). DOI: `10.1007/s00449-016-1557-1` (cit. on pp. 3, 8).

[16] J. Schubert et al. "Bioprocess optimization and control: Application of hybrid modelling". In: *Journal of Biotechnology* 35 (1 1994-06), pp. 51–68. ISSN: 0168-1656. DOI: `10.1016/0168-1656(94)90189-9` (cit. on pp. 3, 8).

[17] V. Galvanauskas, R. Simutis, and A. Lübbert. "Hybrid process models for process optimisation, monitoring and control". In: *Bioprocess and Biosystems Engineering* 26 (6 2004-12), pp. 393–400. ISSN: 16157591. DOI: `10.1007/s00449-004-0385-x` (cit. on p. 3).

[18] B. Bayer et al. "Model Transferability and Reduced Experimental Burden in Cell Culture Process Development Facilitated by Hybrid Modeling and Intensified Design of Experiments". In: 9 (2021), p. 23. DOI: `10.3389/fbioe.2021.740215`. URL: `www.frontiersin.org` (cit. on p. 3).

[19] L. Chen et al. "Hybrid modelling of biotechnological processes using neural networks". In: *Control Engineering Practice* 8 (7 2000-07), pp. 821–827. ISSN: 0967-0661. DOI: `10.1016/S0967-0661(00)00036-8` (cit. on p. 3).

[20] M. Sokolov. "Decision Making and Risk Management in Biopharmaceutical EngineeringOpportunities in the Age of Covid-19 and Digitalization". In: *Cite This: Ind. Eng. Chem. Res* 59 (2020), pp. 17587–17592. DOI: 10.1021/acs.iecr.0c02994. URL: https://dx.doi.org/10.1021/acs.iecr.0c02994 (cit. on p. 3).

[21] H. Narayanan, M. Sponchioni, and M. Morbidelli. "Integration and digitalization in the manufacturing of therapeutic proteins". In: *Chemical Engineering Science* 248 (2022-02), p. 117159. ISSN: 0009-2509. DOI: 10.1016/J.CES.2021.117159 (cit. on p. 3).

[22] M. Sokolov et al. "Hybrid modeling — a key enabler towards realizing digital twins in biopharma?" In: *Current Opinion in Chemical Engineering* 34 (2021-12), p. 100715. ISSN: 2211-3398. DOI: 10.1016/J.COCHE.2021.100715 (cit. on p. 3).

[23] D. C. Psichogios and L. H. Ungar. *A Hybrid Neural Network-First Principles Approach to Process Modeling* (cit. on pp. 3, 8).

[24] M. L. Thompson and M. A. Kramer. *Modeling Chemical Processes Using Prior Knowledge and Neural Networks* (cit. on pp. 3, 8).

[25] N. Sharma and Y. A. Liu. "A hybrid science-guided machine learning approach for modeling chemical processes: A review". In: *AIChE Journal* 68.5 (2022-02). ISSN: 1547-5905. DOI: 10.1002/aic.17609. URL: http://dx.doi.org/10.1002/aic.17609 (cit. on pp. 3, 7, 8).

[26] *[2206.03451] Combining physics-based and data-driven techniques for reliable hybrid analysis and modeling using the corrective source term approach.* URL: https://ar5iv.labs.arxiv.org/html/2206.03451 (cit. on p. 3).

[27] M. Hucka et al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models". In: *Bioinformatics* 19.4 (2003), pp. 524–531 (cit. on p. 4).

[28] N. L. Novère et al. "BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems". In: *Nucleic Acids Research* 34 (Database issue 2006-01), p. D689. ISSN: 13624962. DOI: 10.1093/NAR/GKJ092. URL: /pmc/articles/PMC1347454//pmc/articles/PMC1347454/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC1347454/ (cit. on p. 4).

[29] J. Pinto et al. "SBML2HYB: a Python interface for SBML compatible hybrid modeling". In: *Bioinformatics* 39.1 (2023-01), btad044. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad044. eprint: https://academic.oup.com/bioinformatics/article-pdf/39/1/btad044/49002713/btad044.pdf. URL: https://doi.org/10.1093/bioinformatics/btad044 (cit. on pp. 4, 21).

[30] M. A. Branch, T. F. Coleman, and Y. Li. "A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems". In: *SIAM Journal on Scientific Computing* 21.1 (1999), pp. 1–23. DOI: 10.1137/S1064827595289108. eprint: https://doi.org/10.1137/S1064827595289108. URL: https://doi.org/10.1137/S1064827595289108 (cit. on pp. 5, 28).

[31] R. H. Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: 10.1137/0916069. eprint: https://doi.org/10.1137/0916069. URL: https://doi.org/10.1137/0916069 (cit. on pp. 5, 28).

[32] M. Lalee, J. Nocedal, and T. Plantenga. "On the Implementation of an Algorithm for Large-Scale Equality Constrained Optimization". In: *SIAM Journal on Optimization* 8.3 (1998), pp. 682–706. DOI: 10.1137/S1052623493262993. eprint: https://doi.org/10.1137/S1052623493262993. URL: https://doi.org/10.1137/S1052623493262993 (cit. on pp. 5, 28).

[33] Y Xiang et al. "Generalized simulated annealing algorithm and its application to the Thomson model". In: *Physics Letters A* 233.3 (1997), pp. 216–220. ISSN: 0375-9601. DOI: https://doi.org/10.1016/S0375-9601(97)00474-X. URL: https://www.sciencedirect.com/science/article/pii/S037596019700474X (cit. on pp. 5, 29).

[34] R. Agharafeie et al. "From Shallow to Deep Bioprocess Hybrid Modeling: Advances and Future Perspectives". In: *Fermentation* 9 (10 2023-10), p. 922. ISSN: 23115637. DOI: 10.3390/FERMENTATION9100922/S1. URL: https://www.mdpi.com/2311-5637/9/10/922/htmhttps://www.mdpi.com/2311-5637/9/10/922 (cit. on p. 7).

[35] A. P. Teixeira et al. "Hybrid elementary flux analysis/nonparametric modeling: Application for bioprocess control". In: *BMC Bioinformatics* 8 (1 2007-01), pp. 1–15. ISSN: 14712105. DOI: 10.1186/1471-2105-8-30/FIGURES/8. URL: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-30 (cit. on p. 7).

[36] J. Pinto et al. "A general deep hybrid model for bioreactor systems: Combining first principles with deep neural networks". In: *Computers Chemical Engineering* 165 (2022-09), p. 107952. ISSN: 0098-1354. DOI: 10.1016/J.COMPCHEMENG.2022.107952 (cit. on pp. 7–9, 13, 19).

[37] M. von Stosch et al. "Modelling biochemical networks with intrinsic time delays: A hybrid semi-parametric approach". In: *BMC Systems Biology* 4 (1 2010-09), pp. 1–13. ISSN: 17520509. DOI: 10.1186/1752-0509-4-131/FIGURES/6. URL: https://bmcsystbiol.biomedcentral.com/articles/10.1186/1752-0509-4-131 (cit. on p. 7).

[38] J. Pinto et al. "A General Hybrid Modeling Framework for Systems Biology Applications: Combining Mechanistic Knowledge with Deep Neural Networks under the SBML Standard". In: *AI* 4.1 (2023), pp. 303–318. ISSN: 2673-2688. DOI: 10.3390/ai4010014. URL: https://www.mdpi.com/2673-2688/4/1/14 (cit. on p. 7).

[39] S. Kurz et al. "Hybrid modeling: towards the next level of scientific computing in engineering". In: *Journal of Mathematics in Industry* 12 (1 2022-12), pp. 1–12. ISSN: 21905983. DOI: 10.1186/S13362-022-00123-0/TABLES/1. URL: https://mathematicsinindustry.springeropen.com/articles/10.1186/s13362-022-00123-0 (cit. on p. 8).

[40] X. J. Fan et al. "Nitrification and mass balance with a membrane bioreactor for municipal wastewater treatment". In: vol. 34. IWA Publishing, 1996, pp. 129–136. DOI: 10.1016/0273-1223(96)00502-1 (cit. on p. 9).

[41] J. B. Butt. "Butt-J.B—-Reaction-Kinetics-and-Reactor-Design-2nd-Edition". In: (1980) (cit. on p. 10).

[42] A. Cornish-Bowden. "One hundred years of Michaelis–Menten kinetics". In: *Perspectives in Science* 4 (2015-03), pp. 3–9. ISSN: 2213-0209. DOI: 10.1016/J.PISC.2014.12.002 (cit. on p. 10).

[43] K. Han and O. Levenspiel. *Extended Monod Kinetics for Substrate, Product, and Cell Inhibition*. 1987 (cit. on p. 10).

[44] W. W. Cleland. *ENZYME KINETICSl Further ANNUAL REVIEWS*. 1967. URL: www.annualreviews.org (cit. on p. 10).

[45] F. Consolo et al. *A Computational Model for the Optimization of Transport Phenomena in a Rotating Hollow-Fiber Bioreactor for Artificial Liver*. 2009 (cit. on p. 10).

[46] S. Tebbani et al. "Model-based versus model-free control designs for improving microalgae growth in a closed photobioreactor: Some preliminary comparisons". In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. IEEE, 2016. DOI: 10.1109/med.2016.7535870. URL: https://doi.org/10.1109%2Fmed.2016.7535870 (cit. on p. 10).

[47] B. Gulcan et al. *Optimization Models for Integrated Biorefinery Operations*. 2021. arXiv: 2101.03098 [math.OC] (cit. on p. 10).

[48] R. T. Alqahtani, M. I. Nelson, and A. L. Worthy. "A fundamental analysis of continuous flow bioreactor models governed by Contois kinetics. IV. Recycle around the whole reactor cascade". In: *Chemical Engineering Journal* 218 (2013-02), pp. 99–107. ISSN: 1385-8947. DOI: 10.1016/J.CEJ.2012.12.022 (cit. on p. 10).

[49] M. SHULER. "INVITED REVIEW ON THE USE OF CHEMICALLY STRUCTURED MODELS FOR BIOREACTORS". In: *Chemical Engineering Communications* 36.1-6 (1985), pp. 161–189. DOI: 10.1080/00986448508911252. eprint: https://doi.org/10.1080/00986448508911252. URL: https://doi.org/10.1080/00986448508911252 (cit. on p. 10).

[50] J. Fontbona et al. "Stochastic modeling and control of bioreactors". In: *IFAC-PapersOnLine* 50 (1 2017-07), pp. 12611–12616. ISSN: 2405-8963. DOI: 10.1016/J.IFACOL.2017.08.2203 (cit. on p. 10).

[51] B. Bayer, G. Striedner, and M. Duerkop. "Hybrid Modeling and Intensified DoE: An Approach to Accelerate Upstream Process Characterization". In: (2020). DOI: 10.1002/biot.202000121. URL: https://doi.org/10.1002/biot.202000121 (cit. on p. 11).

[52] M. S. F. Bangi and J. S. I. Kwon. "Deep hybrid modeling of chemical process: Application to hydraulic fracturing". In: *Computers Chemical Engineering* 134 (2020-03), p. 106696. ISSN: 0098-1354. DOI: 10.1016/J.COMPCHEMENG.2019.106696 (cit. on p. 11).

[53] O. Delalleau and Y. Bengio. "Shallow vs. Deep Sum-Product Networks". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/8e6b42f1644ecb1327dc03ab345e618b-Paper.pdf (cit. on p. 12).

[54] H. Mhaskar and T. Poggio. *Deep vs. shallow networks : An approximation theory perspective*. 2016. arXiv: 1608.03287 [cs.LG] (cit. on p. 12).

[55] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 12).

[56] P. Schäfer et al. "The Potential of Hybrid Mechanistic/Data-Driven Approaches for Reduced Dynamic Modeling: Application to Distillation Columns". In: *Chemie Ingenieur Technik* 92 (12 2020-12), pp. 1910–1920. ISSN: 1522-2640. DOI: 10.1002/CITE.202000048. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/cite.202000048https://onlinelibrary.wiley.com/doi/abs/10.1002/cite.202000048https://onlinelibrary.wiley.com/doi/10.1002/cite.202000048 (cit. on pp. 12, 13).

[57] M. Alber et al. "Integrating machine learning and multiscale modeling-perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences". In: (). DOI: 10.1038/s41746-019-0193-y. URL: https://doi.org/10.1038/s41746-019-0193-y (cit. on p. 12).

[58] URL: https://www.aspentech.com/ (cit. on pp. 17, 19).

[59] URL: https://docs.novasign.at/toolbox/index.html (cit. on pp. 17, 19).

[0] K. Merkelbach et al. "HybridML: Open source platform for hybrid modeling". In: *Computers and Chemical Engineering* 160 (2022-04). ISSN: 00981354. DOI: 10.1016/j.compchemeng.2022.107736 (cit. on pp. 18, 19).

[60] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 18).

[61] J. A. E. Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4 (cit. on p. 18).

[62] J. Pinto et al. "SBML2HYB: a Python interface for SBML compatible hybrid modeling". In: *Bioinformatics (Oxford, England)* 39 (1 2023-01). ISSN: 13674811. DOI: 10.1093/bioinformatics/btad044 (cit. on p. 18).

[63] C. Chassagnole et al. *Control of the threonine-synthesis pathway in Escherichia coli : a theoretical and experimental approach*. 2001, pp. 433–444 (cit. on pp. 18, 40, 41).

[64] S. Park and W. F. Ramirez'. *Effect of Transcription Promoters on the Optimal Production of Secreted Protein in Fed-Batch Reactors*. 1990, pp. 1–8 (cit. on pp. 18, 40, 41).

[65] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG]. URL: https://arxiv.org/abs/2109.14545 (cit. on p. 26).

[66] C. Olah. *Understanding LSTMs*. Accessed: 2024-07-08. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. on pp. 26, 27).

[67] Paperspace. *Activation Function*. Accessed: 2024-07-08. n.d. URL: https://machine-learning.paperspace.com/wiki/activation-function (cit. on p. 27).

[68] *Weight Initialization for Deep Learning Neural Networks - MachineLearningMastery.com*. URL: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/ (cit. on p. 27).

[69] C. Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Trans. Math. Softw.* 23.4 (1997), 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236 (cit. on p. 28).

[70] R. H. Byrd, M. E. Hribar, and J. Nocedal. "An Interior Point Algorithm for Large-Scale Nonlinear Programming". In: *SIAM Journal on Optimization* 9.4 (1999), pp. 877–900. DOI: 10.1137/S1052623497325107. eprint: https://doi.org/10.1137/S1052623497325107. URL: https://doi.org/10.1137/S1052623497325107 (cit. on p. 28).

[71] C. Tsallis. "Possible generalization of Boltzmann-Gibbs statistics". In: *Journal of Statistical Physics* 52 (1988-07), pp. 479–487. DOI: 10.1007/BF01016429 (cit. on p. 29).

[72] C. Tsallis and D. A. Stariolo. "Generalized simulated annealing". In: *Physica A: Statistical Mechanics and its Applications* 233.1 (1996), pp. 395–406. ISSN: 0378-4371. DOI: https://doi.org/10.1016/S0378-4371(96)00271-3. URL: https://www.sciencedirect.com/science/article/pii/S0378437196002713 (cit. on p. 29).

[73] Y. Xiang and X. G. Gong. "Efficiency of generalized simulated annealing". In: *Phys. Rev. E* 62 (3 2000), pp. 4473–4476. DOI: 10.1103/PhysRevE.62.4473. URL: https://link.aps.org/doi/10.1103/PhysRevE.62.4473 (cit. on p. 29).

[74] Y. Xiang et al. "Generalized Simulated Annealing for Global Optimization: The GenSA Package". In: *The R Journal* 5.1 (2013), pp. 13–28. DOI: 10.32614/RJ-2013-002. URL: https://doi.org/10.32614/RJ-2013-002 (cit. on p. 29).

[75] K. M. Mullen. "Continuous Global Optimization in R". In: *Journal of Statistical Software* 60.6 (2014), 1–45. DOI: 10.18637/jss.v060.i06. URL: https://www.jstatsoft.org/index.php/jss/article/view/v060i06 (cit. on p. 29).

[76] D. P. Kingma and J. L. Ba. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION* (cit. on p. 29).

[77] T. Tieleman and G. Hinton. *Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning*. Tech. rep. Technical report, 2012. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (cit. on p. 29).

[78] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html (cit. on p. 29).

[79] P. Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on p. 29).

[80] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 29).

[81] URL: https://react.dev/ (cit. on p. 31).

[82] URL: https://mui.com/ (cit. on p. 31).

[83] J. Coene. *firebase: Integrates 'Google Firebase' Authentication Storage, and 'Analytics' with 'Shiny'*. R package version 1.0.2, 2023. URL: https://github.com/johncoene/firebase (cit. on p. 31).

[84] M. Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018 (cit. on p. 31).

[85] M. Jackson. *React Dom*. 2019. URL: `https://github.com/remix-run/react-router` (cit. on p. 32).

[86] URL: `https://www.esbes2024.org/` (cit. on p. 55).