BioMeld

Project:

BioMeld

Grant Agreement (GA) No. 101070328

"A MODULAR FRAMEWORK FOR DESIGNING AND PRODUCING BIOHYBRID MACHINES"

Call: HORIZON-CL4-2021-DIGITAL-EMERGING-01
Type of action: Research and Innovation action (RIA)
Start date of project: 01/10/2022 Duration: 36 months

D2.1: THE FRAMEWORK SPECIFICATIONS

DELIVERABLE FACTSHEET

Project title | Acronym | Number		A Modular Framework for Designing and Producing Biohybrid Machines | BioMeld | 101070328	
Due Date:	31/03/2023	**Date of submission:**	29/03/2023
Month of Project	6	**Month of submission:**	
Title of deliverable:	D2.1: The Framework Specifications	**Work Package:**	WP2 – Modelling and simulation framework
Dissemination level:	PU	**Version/Status**	V1
Deliverable leader (Name|Organisation)	UNSPF	**Editor(s)**	UNSPF – Milos Savic, Vladimir Kurbalija, Igor Balaz
Contribution of partners	UNSPF wrote the initial version. UNICA, IBEC, and UWE contributed to certain sections. UNSPF finalized the Deliverable.		
Final review and approval	Entire consortium		
Keywords	Biohybrid machines, living cell actuators; artificial materials; simulation; modular framework; AI algorithms; scalable manufacturing; 3D voxel-based evolvable simulator;		

	observable parameters; physico-chemical simulation modules; optimization; machine learning; deep learning; data exchange formats; validation; verification; control software; simulation feedback loop.
Abstract	The BioMeld project aims to develop a modular modeling and simulation framework for the digital design of biohybrid machines (BHMs) by leveraging AI-powered algorithms. The framework will enable the efficient manufacturing of BHMs by refining BHM designs through a series of simulation modules. The project's proof-of-principle is to develop a biohybrid catheter as a medical device capable of reaching hard-to-reach regions of the human body to release drugs. The deliverable outlines the methodological foundations, architecture, and plans for validation and verification of the framework.

Document change history		
Date	**Authors**	**Description**
29/01/2023	Milos Savic, Vladimir Kurbalija, Igor Balaz	Document outline created
19/03/2023	Milos Savic, Vladimir Kurbalija, Igor Balaz	Final version created and sent to all participants for review

CONSORTIUM

	Name	Short Name	Country
1.	UNIVERZITET U NOVOM SADU, POLJOPRIVREDNI FAKULTET NOVI SAD	UNSPF	Serbia
2.	SCUOLA SUPERIORE DI STUDI UNIVERSITARI E DI PERFEZIONAMENTO S ANNA	SSSA	Italy
3.	FUNDACIO INSTITUT DE BIOENGINYERIA DE CATALUNYA	IBEC-CERCA	Spain
4.	SMART SENSING S.R.L.	SMART SENSING	Italy
5.	UNIVERSITA DEGLI STUDI DI CAGLIARI	UNICA	Italy
6.	LEVERETTE LANCE	Lance Leverette	Belgium
7.	The University of the West of England	UWE Bristol	United Kingdom

EXECUTIVE SUMMARY

The BioMeld project aims to develop a modular modeling and simulation framework for the digital design of biohybrid machines (BHMs), which combine living cell actuators with artificial materials to achieve greater autonomy, flexibility, and energy efficiency compared to standard robots. This framework will enable the use of demanding AI algorithms in early stages of BHM manufacturing to find the most suitable and efficient BHM designs, as well as scale-up BHM manufacturing by incorporating feedback from experimental analysis of fabricated BHMs into simulation scenarios, thus reducing error-prone manual

steps. The project plans to use the framework to develop a biohybrid catheter as an innovative medical device to release drugs in hard-to-reach regions of the human body. The deliverable includes a presentation of the methodological foundations of the BHM modeling and simulation approach, the architecture of the BioMeld software pipeline, plans and directions for the validation and verification of the framework, and the elaboration of control software to enable the simulation feedback loop for BHM design.

Section 2 of the report presents the methodological foundations of the Biohybrid machines (BHM) modeling and simulation approach. This approach is based on a 3D voxel-based evolvable simulator (Section 2.1), estimation of observable parameters (Section 2.2), physico-chemical simulation modules (Section 2.3), and the optimization of BHM actuator parameters using machine/deep learning techniques (Section 2.4). In Section 3, the architecture of the modeling and simulation framework for the BioMeld project is described. This includes the broad architecture of the BioMeld software pipeline in Section 3.1, individual modules of the proposed pipeline in more detail in Section 3.2, and data exchange formats among modules in Section 3.3. Section 4 outlines the plans and directions for the validation and verification of the framework, while Section 5 elaborates on control software that will enable the simulation feedback loop for BHM design. Finally, the last section provides the conclusion of the deliverable.

LEGAL NOTICE
This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement number 101070328.
Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

None

LIST OF ABBREVIATIONS

Abbreviation	Description
BHM	Biohybrid machine
NN	Neural network
DNN	Deep neural network
MLP	Multilayer perceptron
MAE	Mean absolute error
MSE	Mean squared error
SCC	Strongly connected component
FEM	Finite element method
PDEs	Partial Differential Equations
FSI	Fluid-structure interaction
GA	Genetic algorithms
ANN	Artificial Neural Networks
NEAT	Neuroevolution of Augmented Topologies

1 INTRODUCTION

Biohybrid machines (BHMs) combine living cell actuators with artificial materials in order to achieve greater autonomy, flexibility and energy efficiency compared to standard robots. Although BHMs have great promise for future applications in many fields, BHM technologies are still in early stages of development with no significant efforts made towards efficient BHM manufacturing leveraging AI-powered modeling and simulation of BHMs. One of the objectives of the BioMeld project is to develop a modeling and simulation framework for the digital design of BHMs. This framework is envisioned as a modular framework in which BHM designs are refined through a series of simulation modules arranged according to their level of abstraction, from the most abstract to the most detailed simulation modules. This modular approach will enable us: (1) to couple demanding AI algorithms in early stages of BHM manufacturing in order to find the most suitable, robust and efficient BHM designs, and (2) to scale-up BHM manufacturing by incorporating feedback obtained from experimental analysis of fabricated BHMs into simulation scenarios, thus significantly reducing error-prone manual steps. As a proof-of-principle, we will use the framework to develop a biohybrid catheter as an innovative medical device able to arrive into hard-to-reach regions of the human body in order to release drugs there.

The rest of this deliverable is organized as follows. In Section 2 we present methodological foundations for our BHM modeling and simulation approach that is based on a 3D voxel-based evolvable simulator (Section 2.1), estimation of observable parameters (Section 2.2), physico-chemical simulation modules (Section 2.3) and optimization of BHM actuator parameters using machine/deep learning techniques (Section 2.4). The architecture of the modeling and simulation framework that will be developed in the BioMeld project is explained in Section 3: Section 3.1. presents the broad architecture of the BioMeld software pipeline, Section 3.2 describe individual modules of the proposed pipeline in more details, while Section 3.3 describes data exchange formats among modules. In Section 4 we outline our plans and directions for the validation and verification of the framework. Section 5 elaborates on control software that will enable the simulation feedback loop for BHM design. The last section concludes the deliverable.

2 MODELING METHODOLOGY

This section will present necessary technologies needed for modeling the framework for designing BHMs. The first important concept is the generation of potential morphologies of BHMs and their environment. This generation is performed using the Voxelyze modeling framework and by applying genetic algorithms and compositional pattern producing networks on candidate morphologies. Since the number of model parameters (of BHMs) is expected to be huge, an estimation and selection of observable parameters is an important concept here. This task will be realized using the theory of complex networks. After successful initial design, the BHMs should be tested in a particular physical and chemical environment which includes: examination of electric stimulus-response, rotation and bending dynamics, etc. The COMSOL Multiphysics software is an appropriate tool for this testing environment. Finally, the optimization of actuators in BHMs is performed using AI approaches, and it will consist of three deep neural networks: Simulation DNN, Actuator DNN and Policy DNN. All presented concepts will be discussed in the following sections, and their concrete usage will be given in the architecture pipeline.

2.1 3D VOXEL-BASED EVOLVABLE SIMULATOR

The first task in the Modeling and simulation framework of the BioMeld project is the design of BHM and its morphologies. The Voxelyze physics engine simulates BHM design and deals with parameters related to physical properties of materials used to build BHM. Additionally, the morphology generator produces candidate BHM morphologies (voxel-based geometries), as well as other objects needed for simulation (e.g., vascular walls). This intelligent generation of potential morphologies is based on Genetic algorithms and on compositional pattern producing networks.

2.1.1. BIOMELD-VOXELYZE

Voxelyze is a highly sophisticated physics engine designed to model and simulate the mechanical behavior of soft and flexible materials (Hiller et al. 2014). The system uses a finite element method, where a material is divided into a mesh of interconnected elements, to model the deformation and mechanical properties of the material. The elements in the mesh represent the behavior of small volumes of the material, and their interactions with each other are used to describe the overall mechanical response of the material.

In addition to its sophisticated modeling capabilities, Voxelyze is also designed to be highly flexible and customizable. This allows researchers and engineers to easily modify the system to study the behavior of different types of materials, and to perform simulations with varying levels of detail and complexity. For example, the system can be used to model the mechanical response of gels, flexible electronics, and biological tissues to various stimuli, such as temperature changes, pressure changes, and applied forces.

The main modeling approach in Voxelyze is based on voxels and mass-spring lattice. Each voxel in the lattice is modeled with six degrees of freedom: three translational and three rotational degrees of freedom. Furthermore, each voxel contains its mass and a rotational inertia to support more complex simulations with lateral shearing and rotation. Finally, each voxel is connected to 6 neighboring voxels with connection elements, where every connection contains its own transnational and rotational stiffness (**Figure 1**). Such a setup provides a very realistic deformation simulation under different forces and moments.

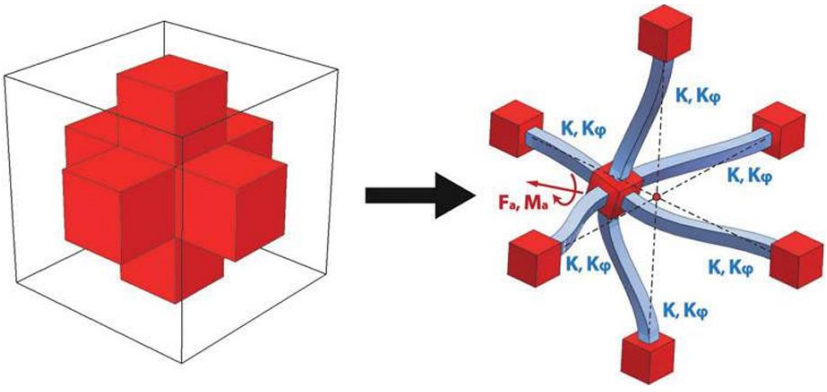

Figure 1. Voxel connection in Voxelyze (Hiller et al. 2014).

Besides basic soft-body simulation features, Voxelyze contains several advanced features which improve its simulation capabilities:

- **Neighboring voxels of different properties**: when two adjacent voxels are composed of different materials, Voxelyze calculates composite properties for elastic modulus and stiffness. This allows easy and qualitative multi-material modeling.
- **Selecting a timestep**: a small timestep is needed to prevent numerical instability, but a higher step is more desirable to be computationally efficient.
- **Damping management**: damping is needed to prevent the accumulation of numerical errors and to provide a realistic oscillation damping properties of materials.
- **Gravity**: since the force is calculated for each voxel, the mass of the voxel is multiplied with the acceleration of gravity and added/subtracted from the vertical component of the force.
- **Floor**: the stiffness of some voxel contacting the floor is the stiffness of the floor on that location. No composite stiffness is calculated in these situations because in that case the stiffness of the floor should be infinite.
- **Friction**: the Coulomb friction model is used to calculate the resistance of the object to motion under the application of the force. When a static friction threshold has been exceeded, the object starts to move in the direction of the force. The movement of the object is reduced using the dynamic coefficient of friction.
- **Collision detection**: collision detection is carefully implemented due to computational efficiency. Naive implementation where each voxel is compared to every other would result in $O(n^2)$ efficiency, which is unacceptable. Instead of that, Voxelyze precomputes the surface voxels, and during the simulation calculates possible collisions only between these surface voxels.
- **Temperature changes and volumetric actuation**: the volume change of the voxels is implemented by changing the length between adjacent voxels when computing the elastic force between them. The volumetric behavior of materials is controlled through coefficient of thermal expansion.

Overall, Voxelyze is a valuable tool for researchers and engineers in a wide range of fields, including materials science, robotics, and biology. By providing a detailed and accurate model of the mechanical behavior of soft materials, Voxelyze can help researchers to better understand the properties of these materials and to design new materials with improved mechanical properties. Moreover, its multi-material modeling capabilities makes Voxelyze a very convenient tool for BHM design within the BioMeld project.

Voxelyze engine will be used to build a simulator (BioMeld-Voxelyze) which will be used for simulation of BHM behavior and for selecting optimal BHM designs. This component will be part of the BioMeld modeling and simulation framework whose main goal is the appropriate design of BHMs. The task of the BioMeld Voxelyze component is to consider and select parameters related to physical properties of materials used to build BHM and its morphology (geometry). More precisely, in Voxelyze engine we will apply force to the BHM (e.g. catheter) to move it through environment which contains other objects (e.g. vascular walls). The Voxelyze will take care of various physical aspects including: forces, collision detection, gravity, material properties (of catheter and vascular walls) and temperature. Moreover, the Morphology generator component will generate different

morphologies of BHMs (catheter) and environmental objects (vascular cells), which should be considered within simulation.

2.1.2. ENCODING OF MORPHOLOGIES USING CPPN

Given that the design of the BHM will have to consider a plethora of parameters on the physics simulator and the fact that the complexity of possible morphologies may prove to be challenging for human intuition and/or engineering skills, an automated generation and optimization process will be utilized. More specifically, variations of Genetic Algorithms (GAs) will be employed to navigate through the search space of parameters and discover the optimal morphology for a BHM catheter. However, in evolutionary computation and artificial intelligence the representation of solutions is a critical factor (Stanley, 2007; Tsompanas et al., 2022). Provided the background of the problem, the encoding by Compositional Pattern Producing Networks (CPPNs) (Stanley, 2007; Cheney et al., 2014) will be used here.

The CPPN encoding is a member in the category of developmental encodings, meaning that the objective of its principles is to implement an abstraction of the process of natural development within the evolutionary algorithm. This encoding is based on the composition of functions that results into the emergence of symmetry, imperfect symmetry, repetition, repetition with variation, preservation of regularities and elaboration of existing regularities; all being important attributes of natural development, while their combinations are capable of producing extremely elaborate and complex patterns. The use of CPPNs have been successfully demonstrated in different fields, such as 2D image (Secretan et al., 2008), 3D objects (Clune & Lipson, 2011) and soft robot design (Cheney et al., 2014). Thus, we will further investigate its applicability to the framework that will assist the design of BHMs.

The formalism of CPPNs is very similar with that of artificial neural networks. However, they are not to be confused, as CPPNs are an abstraction of development, while ANNs are an abstraction of the human brain. Moreover, nodes within ANNs usually have restricted activation functions (i.e. ReLU and/or sigmoid), while in CPPNs the functions can be of any form and characteristics (i.e. Gaussian, sinusoid, exponential and more). On the other hand, similarly to ANNs the connections between nodes are weighted, meaning that the output of a node is multiplied with the weight of the outgoing connection. Also, when more than one connection terminates to the same node, then the input to that node is the addition of the weighted outputs of the previous nodes. It is noteworthy that the topology of CPPNs is not constrained and can implement any possible relationship. Also, CPPN is not limited by the scales of the problem. Namely, the evolutionary optimization on the computationally expensive simulations can be performed in a low resolution of the problem, and then the CPPN can produce a morphology of higher resolution without further optimization rounds.

In previous applications of optimizing a walking soft robot morphology, the input nodes of a CPPN were representing the Cartesian coordinates in a 3D design space of the robot morphology. Whereas, the output of the network was the existence or not of the specific voxel (with coordinates as in the input nodes) and its properties. Namely, if the given voxel will be passive (and more specifically stiff or soft) or active (and more specifically expanding or contracting in phase with the rest of the voxels). A corresponding concept will be utilized as a part of the software design

framework. Thus, the geometry of a BHM will be encoded as a bitstring that will indicate if a material will be present or absent at each lattice point of the workspace and in the case of presence the characteristics of the material (i.e. resting volume, Young's modulus, Poisson's ratio, and coefficients of static and kinetic friction).

Provided that the CPPNs are structurally resembling artificial neural networks, established methodologies that have been used to evolve and optimize the output of neural networks can be employed with small adaptations. For example, a method that is used for training neural networks instead of back-propagation is the Neuroevolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002). With some alternations in the procedure, NEAT can be implemented to assess CPPNs and evolve increasingly complex instances that will be evaluated on their fitness upon the given problem, like a walking soft robot (Cheney et al., 2014). In a CPPN, both the architectures and the weights of connections can alter, by mutating the individuals in the simulated population. For instance, six types of mutations can be defined (add vertex/edge, remove vertex/edge, modify vertex/edge) with similar or biased probabilities. NEAT has proved to be a powerful method based on its principles of complexification, speciation and use of innovation measure; thus, inspired multiple variations that were applied in different problems (Papavasileiou et al., 2021). In this setting the most appropriate variation of NEAT will be implemented as a part of the software framework.

2.2. ESTIMATION OF OBSERVABLE PARAMETERS

When modeling complex systems such as BHMs for analytic, simulation and predictive purposes, it is usually impossible to make an "ideal" model covering every single variable/parameter. Moreover, "large" models (in terms of the number of parameters) require significant time for parameter tuning and analysis, while their simulations may be extremely slow. Additionally, less complex models (models with less free variables) usually provide more accurate predictions compared to more complex models that may be overfitted to particular scenarios. Consequently, we will identify a subset of variables that are essential for monitoring and controlling BHM behavior.

The main characteristic of complex systems is that their variables are usually interdependent. Therefore, a carefully selected subset of variables can provide sufficient information about the rest of variables, allowing us to reconstruct the complete internal state of the BHM, thus making the whole system observable. This subset of variables are also called sensory nodes. To estimate (infer) sensory nodes we took the approach proposed by Lie et al. (2013) that is inspired by the structured system theory in which the interdependence between parameters is represented by an inference diagram structured as a directed graph. The procedure for estimating sensory nodes consists of the following steps:

(1) **Construct inference diagram (directed graph).** The inference diagram is constructed from a set of differential equations describing BHM behavior. Each node in the diagram represents one of the variables appearing in the equations. Two nodes A and B are connected by a directed link B → A if A appears in the differential equation describing the dynamics of B, implying that one can collect information on A by monitoring B as a function of time. In other words, the inference graph actually captures the information flow among interdependent variables in differential equations describing the behavior of the whole system.

(2) **Identify strongly connected components in the inference diagram**. A strongly connected component in a directed graph is its maximal subgraph such that for each two nodes X and Y in the subgraph there is a directed path from X to Y and also a directed path from Y to X. Strongly connected components in the inference diagram actually indicate cyclic dependencies among variables, which implies that the state of any variable within a strongly connected component can be inferred from the state of any other variable belonging to the same strongly connected components. The Tarjan algorithm (Tarjan, 1972) is commonly used to identify strongly connected components in a directed graph and its implementation is present in various programming libraries for processing, drawing and analyzing graphs and complex networks.

(3) **Identification of root strongly connected components and selection of sensory nodes**. From detected strongly connected components we can form a directed graph showing dependencies between them. In this dependency graph, each strongly connected component is represented by one node and two strongly connected components P and Q are connected by a directed link P → Q if P contains a node A and Q contains a node B such that there is a directed link A → B. In other words, links in the dependency graph connect adjacent strongly connected components. A root strongly connected component is a component that has no incoming links in the dependency graph. This means that any node from the root connected component enables observability of all reachable nodes. Thus, sensory nodes are selected from identified root strongly connected components (one or more nodes that are selected either randomly or by their importance quantified by node centrality metrics).

Figure 2 shows an example inference diagram describing a system with 12 variables (denoted by letters from A to L). Its strongly connected components are framed by dashed circles and we can see that there are 5 strongly connected components: two containing one variable, two containing two variables and one encompassing 6 variables. The dependency graph of strongly connected components is shown on the right side. It can be seen that there are two root strongly connected components: {D} and {G, H, I, J, K, L}. Thus, the minimal set of sensory nodes contains D and one node selected from the second root strongly connected component (e.g., G).

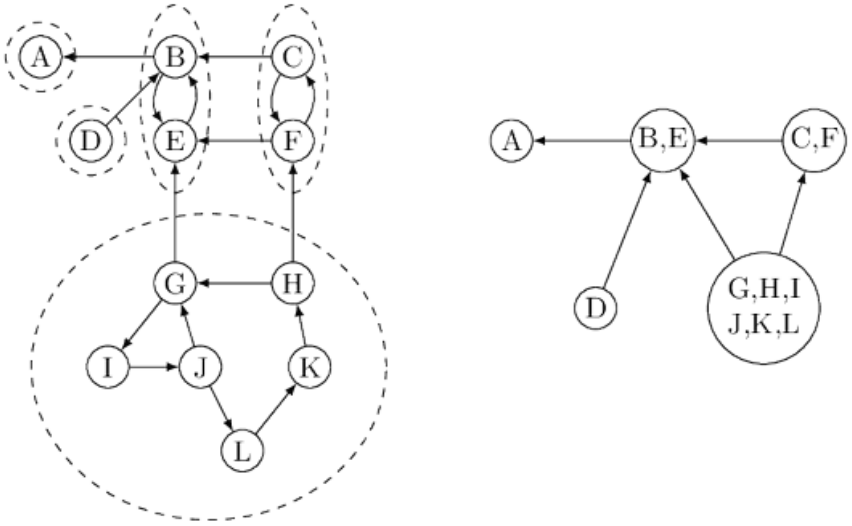

Figure 2. An example inference diagram (left graph), its strongly connected components (dashed circles in the left graph) and the graph of dependencies between strongly connected components (right graph).

As already noted, any node from a root strongly connected component can be selected to be a sensory node. However, we can also rank nodes in the inference graph by computing their structural importance quantified by node centrality metrics (Savić et al., 2019) and then select sensory nodes that have the highest importance. The three commonly used node centrality metrics for directed graphs are:

(1) **Betweenness centrality.** The betweenness centrality of a node quantifies its importance as the probability that it appears on a shortest directed path connecting any two nodes in the graph. This metric is motivated by the fact that the fastest information flow between a pair nodes usually happens via the shortest path connecting them. Thus, nodes on the shortest paths have a vital role for information flow in the whole system.

(2) **Closeness centrality.** The closeness centrality of a node is inversely proportional to the total shortest-path distance between the node and all other nodes in the graph. The intuition behind this metric is that a node can be considered important if it is in proximity to many other nodes.

(3) **PageRank centrality**. This centrality metric is a variant of eigenvector centrality adapted for directed graphs that do not contain exactly one strongly connected component. The main idea of the metric is that the importance of a node depends on the importance of nodes linking to it (its in-neighbors), where each in-neighbor divides its importance into equal shares per out-neighbors. Thus, this metric is recursively defined and it can be computed by iterative approximations starting from the initial state in which all nodes have an equal PageRank.

Identified root components can also be relatively large (for example, the largest connected component in **Figure 2** is root). Large root components can be further decomposed into smaller cohesive subgraphs that are also called communities. A community is a subset of nodes that are more densely connected among themselves than with the rest of the graph. The decomposition into communities can be achieved by a variety of community detection algorithms (Fortunato & Newman, 2022; Jin et al. 2023). Widely used community detection algorithms are greedy modularity optimization, Louvain, InfoMap, Walktrap, Label propagation, etc. Therefore for large root components, we can select sensory nodes such that each community is covered.

2.3. PHYSICO-CHEMICAL MODULES

The COMSOL Multiphysics software uses a finite element method (FEM) approach to solve partial differential equations (PDEs) that describe the behavior of physical systems. The software starts with transport phenomena, electromagnetic field theory, and solid mechanics and allows users to solve their particular simulation needs. Users create a virtual model of a physical system, define its properties and parameters, and simulate its behavior under different conditions. The fluid-structure interaction (FSI) in COMSOL Multiphysics is a multiphysics coupling between the laws that describe

fluid dynamics and structural mechanics. This phenomenon is characterized by stable or oscillatory interactions between a deformable or moving structure and a surrounding or internal fluid flow.

The FSI-COMSOL Multiphysics tool in the line of a self-monitoring and self-controlling manufacturing pipeline of BHMs serves as part of a physico-chemical module to examine the electric stimulus-response, rotation and bending dynamics of BHMs in a physically realistic way. The set of initial BHM parameters are pushed through a COMSOL pipeline to produce physically realistic scenarios which will add more details to initial BHM designs. These simulations will also result in the set of parameters that explains the interaction between the BHM design and environment and the changes in the environment produced by BHM.

The COMSOL simulation pipeline requires: 1) setting up the model environment - 2D or 3D environment and stationary or time dependent simulation, with appropriate physics modules (e.g. electromagnetics, FSI, mechanics); 2) creation of geometrical objects - geometry of the BHMs created by BioMeld-Voxelyze simulation module are imported as object files, while simple 2D or 3D representation of veins and arteries and surrounding tissues BHM created inside the COMSOL graphics module as base model (more complex structures and physically realistic models are imported object files as well); 3) specify material properties - material properties of the base model are introduced through a material library with properties defined in a database like Tissue Properties Database V4.1 (DOI: 10.13099/VIP21000-04-1). The material properties of BHMs will be introduced through a parameter file in .txt format; 4) physics boundary conditions defined in base object will reflect several vasculature scenarios (different vascular diameters, blood pressure, branching architectures); 5) meshing will be performed by COMSOL.

Simulation results are in the form of matrix with value in every mesh point for each simulated time point.

2.4. OPTIMIZATION OF ACTUATOR PARAMETERS

Actuators in BHMs are extremely difficult to model accurately for three reasons: (1) their dynamics involve nonlinear and nonsmooth dissipation, (2) they contain cascaded feedback loops, and (3) they may have a large number of internal states that are not directly observable. Thus, instead of making an analytical model for optimization of actuator parameters we will rely on predictive models based on deep learning techniques to obtain an action-to-torque relationship. Our approach to optimize actuator parameters consists of three deep neural networks (DNN) put in the reinforcement learning loop (**Figure 3**) inspired by a similar approach proposed by Hwangbo et al. (2019):

1. **Simulation DNN**. This DNN is trained from COMSOL simulation results and it predicts the next state of the BHM (position and velocity) for given joint torques and the current state.
2. **Actuator DNN**. This DNN is trained from real (experimental) data and it predicts torques for given command actions. More precisely, the actuator network outputs an estimated torque at the joints given a history of position errors (the actual position subtracted from the commanded position) and velocities. Additionally, we assume that the dynamics of the actuators are independent of each other which means that one Actuator DNN model is trained for each actuator separately.
3. **Policy DNN**. This DNN is trained in the reinforcement learning loop and it predicts command actions for a given BHM state.

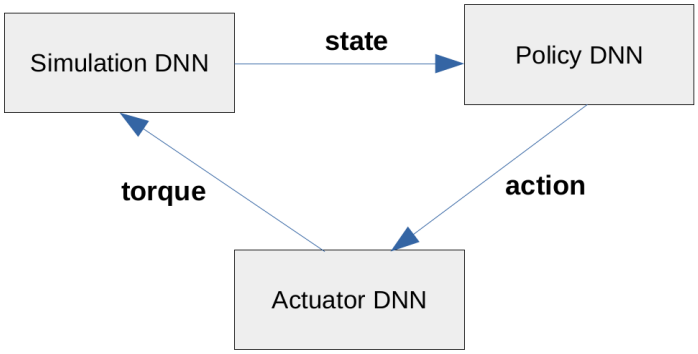

Figure 3. Optimization of actuator parameters by three deep neural networks in the reinforcement learning loop.

Simulation, Actuator and Policy DNNs can be realized as feed-forward multi-layer perceptron (MLP) neural networks (Goodfellow et al., 2016). Such neural networks consist of a sequence of layers. All neurons from the *k*-th layer are connected to all neurons from the (*k*+1)-th layer via weighted edges. The weights of neural network edges together with biases associated to each node constitute the set of real-valued model parameters that are learned on a given training dataset. The nodes in the first layer are input values for a feed-forward mechanism of the neural network, the last layer contains neural network output values (predictions for input values), while nodes in all other intermediate layers can be considered as hidden variables through which input values are transformed to obtain output values. Let us assume that the training dataset is composed of data points in the form <*X*, *y*> where *X* is the feature vector of input variables and *y* is the feature vector of output variables. Then, the feed-forward mechanism of MLP for an arbitrary data point p can be described by the following recursive equations:

$$x_i^{(l)}(p) = W_i^{(l)} y^{(l-1)}(p) + b_i^{(l-1)}$$
$$y_i^{(l)}(p) = \sigma\left(x_i^{(l)}(p)\right)$$
$$y_i^{(0)}(p) = X_i(p)$$

where upper scripts in brackets denote the layer index, $X_i(p)$ is the value of *i*-th feature in *X* for *p*, W_i is the vector of neural network weights for *i*-th layer, b_i is the bias associated to the *i*-th neuron, x_i and y_i are the input and output value to *i*-th neuron, respectively, and σ denotes the activation function (e.g., sigmoid, ReLU, softsign, tanh, etc.). Initially, we will use two activation functions - tanh and softsign. Softsign activation function is computationally efficient and provides a smooth mapping, while tanh is more suitable for nonlinear systems and yields less aggressive trajectories when subjected to disturbances.

The model parameters (weights and biases) are learned by minimizing a loss function on the training dataset. The loss function quantifies the difference between neural network output values for instances in the training data and their real *y* values (that are also given in the training data). The loss function is minimized in a given number of epochs *E* (one epoch is one learning round considering all instances in the training data), where model parameters are updated after processing batches each

containing *B* instances (this value is known as batch size). Learning hyperparameters *E* and *B* are specified during neural network training (typical values for *E* and *B* are 100 and 32, respectively).

The architecture of a neural network is determined by the number of layers and the number of neurons per layer, but also by the type of predictive problem solved by the neural network. Simulation and Actuator DNNs deal with regressions (predicting numerical values). In such cases, the number of nodes in the output layer is equal to the number of numerical values predicted, and those nodes are activated by the linear activation function. For determining model parameters, typically used loss functions are the mean squared error (MSE) and the mean absolute error (MAE):

$$MAE = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n}$$

$$MSE = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}$$

where n represents the number of data points in the training dataset, while x_i and y_i are vectors of input and output values for i-th data point. The selected loss function is typically optimized by some gradient-based optimizer such as Adam, AdaGrad or RMSProp (Goodfellow et al., 2016). The same functions are used to evaluate predictive accuracy of the model on independent test dataset (i.e., data points which are not used for training the predictive model).

We represent the policy learning problem for BHMs in discretized time. At every time step, the agent (BMH controller) obtains an observation (measurements of BHM states), performs an action (position commands to actuators) and achieves a scalar reward. The aim is to find a policy that maximizes the discounted sum of rewards. The rewards are specified so as to induce the behavior of interest. A variety of reinforcement learning algorithms can be applied to the specified policy optimization problem. Initially, we will test Trust Region Policy Optimization (Schulman et al., 2015), a policy gradient algorithm that has been demonstrated to learn locomotion policies in simulation (Schulman et al., 2016).

3 ARCHITECTURE OF THE MODELING AND SIMULATION FRAMEWORK

The main goal of the BioMeld modeling and simulation framework is to provide a software pipeline for the digital design of BHMs. In this pipeline, BHM designs are refined through a set of software modules arranged according to their level of abstraction and increased constraints.

The most abstract modules identify the most important parameters enabling the observability of the whole BHM system and produce a set of primary BHM morphologies taking into account soft-body constraints (physical properties of materials) and desired behavior under moving conditions (stable geometries with low pressure on environmental boundaries, e.g. vascular walls, low sensitivity to small perturbations in actuator parameters and minimization of actuating BHM parts). The primary BHM morphologies with the set of observable parameters are then forwarded to a chain of physico-chemical modules each adding more constraints (e.g., response to electric stimulus, rotation and bending dynamics) for more detailed simulations of initial BHM designs.

Detailed physico-chemical simulations result with the first-generation BHM design that is fabricated and experimentally evaluated. The results of those evaluations together with the results of physico-chemical simulations are then analyzed using machine learning techniques to optimize BHM actuator parameters leading to improved next-generation BHM designs.

3.1. ARCHITECTURE OF THE BIOMELD SOFTWARE PIPELINE

The architecture of the BioMeld software pipeline is shown in **Figure 4**. It could be seen that the pipeline has a three-layer architecture. The first layer contains two software modules: Estimator of observable parameters and BioMeld Voxelyze. The estimator module determines an optimal set of observable BHM parameters that are used in physico-chemical simulations. On the other hand, BioMeld Voxelyze (evolutionary voxel-based simulator) deals with parameters related to physical properties of materials used to build BHM and its morphology (geometry). By utilizing evolutionary AI algorithms, this simulator produces primary BHM candidates that are further simulated by physico-chemical modules.

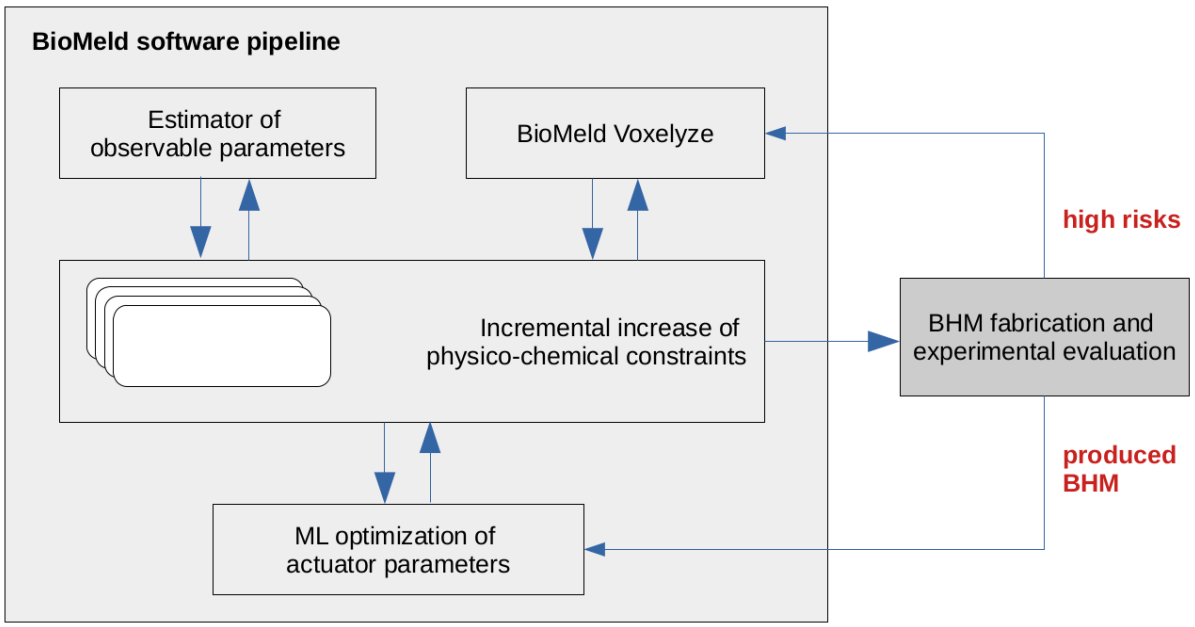

Figure 4. The architecture of the BioMeld software pipeline. Down blue arrows denote data-flow relations, while up blue arrows denote feedback relations between different modules in the pipeline.

The physico-chemical modules in the second layer of the pipeline incrementally increase physico-chemical constraints into performed BHM simulations. There is a direct feedback loop from the second to the first layer. In case that physico-chemical simulations reveal unstable BHM designs under their constraints then BioMeld-Voxelyze simulations are repeated by narrowing their parameter space. Similarly, if physico-chemical simulations identify that selected sensory nodes do not sufficiently capture the controllability of BHM then the estimator is rerun on an expanded set of differential equations describing BHM behavior.

The second layer of the pipeline results with the BHM design for fabrication. This design is first pre-evaluated according to the manufacturing constraints and uncertainties. In case of high risks the design is discarded with the feedback to the BioMeld software pipeline regarding ranges of physical properties of materials that should be used. In case of the approved BMH design, the corresponding BHM is fabricated and experimentally evaluated in the same scenarios that are used in physico-chemical simulations.

The last layer of the pipeline contains machine learning software modules that optimize BHM actuator parameters from simulation data and BHM experimental evaluation data. This layer implements three deep neural networks (DNN): simulation DNN, actuator DNN, policy DNN. Those three networks are mutually coupled in a reinforcement learning loop: (1) simulation DNN is trained from simulation data, (2) actuator DNN is trained from real data (BHM experimental evaluations) and (3) policy DNN is trained in the reinforcement learning loop to learn actions needed to reach a desired goal (e.g., actions to move a BHM catheter through a vascular system to a desired position).

3.2. BIOMELD SOFTWARE MODULES

The module for **estimating observable parameters (EOP)** selects a subset of parameters enabling the controllability of the BHM from a set of differential equations describing its behavior. This component will be realized in the Python programming language. It will consist of two subcomponents: EOP parser and EOP graph analyzer (**Figure 5**). The EOP parser reads a textual input file containing the description of relationships in the set of differential equations and forms its graph representation. Textual input files will be structured according to the following formal grammar:

```
EQUATION_SET->(EQUATION"\n")+    EQUATION->   VARIABLE   "#"   DEPENDENCIES
VARIABLE              ->   ["a"  ..  "z"|  "A"  ..  "Z"  |  "0"  ..  "9"]+
DEPENDENCIES  -> VARIABLE | VARIABLE "," DEPENDENCIES
```

where "\n" denotes new line, + is the operator meaning one or more occurrences, | is the or operator and EQUATION_SET is the starting non-terminal symbol. In other words, each line of an input file corresponds to one differential equation and contains character "#" separating the variable from the left side of the equation and its dependencies contained from the right side. Since the formal grammar is relatively simple, the parser for it will be coded from scratch without relying on any parser generator (i.e., so-called hand-made parsers). The output of the parser will be an intermediate form structured as a graph. For this purpose we will use the NetworkX library (Hagberg et al, 2008). This library also contains the implementation of the Tarjan algorithm for detecting strongly connected components that will be part of EOP graph analyzer. After detecting strongly connected components (SCCs), the EOP graph analyzer will create a reduced graph representation in which: (1) each node corresponds to one SCC, and (2) two SCCs A and B are connected by a directed link A → B if A contains a node connected by an outgoing link to a node in B. Root SCCs can be trivially selected from the reduced representation as nodes without incoming links. In case of large root SCCs we will use community detection algorithms to further decompose it into smaller parts and select the most important nodes by using node centrality metrics. The NetworkX library will be

used to compute node centrality metrics. For community detection we will use existing implementations in NetworkX, but also in the CDLIB library (Rossetti et al, 2019) which provides a significantly larger number of community detection methods. The selected sensory nodes will be then exported in an output textual file, one sensory node per line.

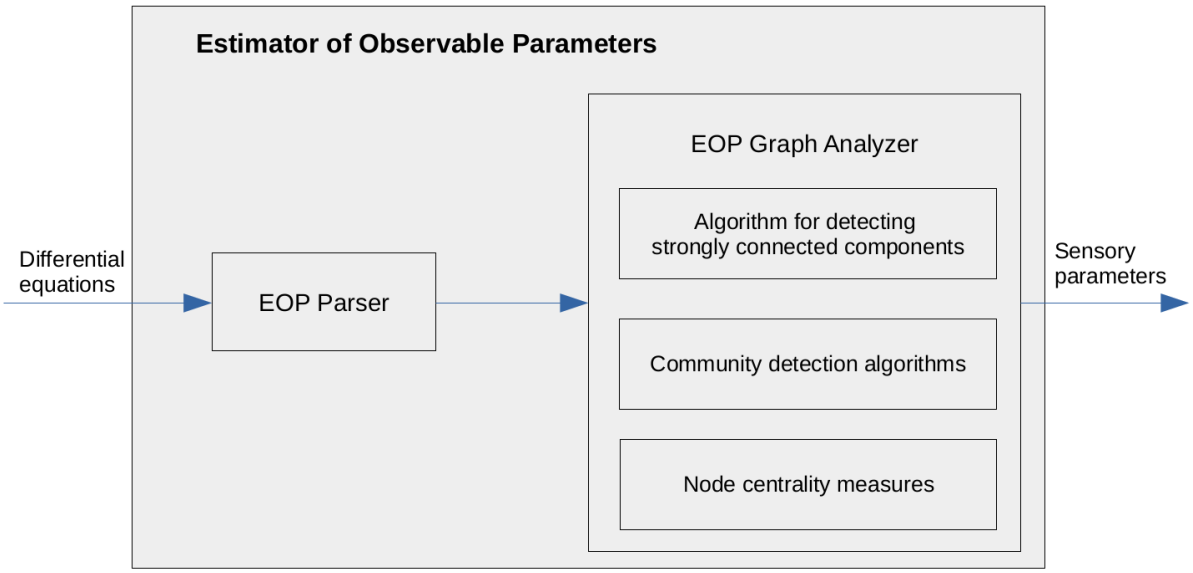

Figure 5. The architecture of the module for estimating observable parameters. Blue arrows denote data-flow relations between different components in the module.

BioMeld-Voxelyze is a voxel-based evolvable simulator of BHMs. In this simulator, a BHM (e.g., catheter) is represented as a set of adjacent voxels in a 3D space to which a force can be applied enabling BHM movements in an environment containing other objects (e.g., vascular walls). It is designed to contain the following software components (**Figure 6**):

- **Voxelyze Engine**. This component is a wrapper around the Voxelyze library (Hiller et al. 2014) implementing soft-body physics including movements stimulated by forces applied on voxelized objects, collision detection, gravity and temperature effects.
- **Morphology Generator**. This component is responsible for producing candidate BHM morphologies (voxel-based geometries), as well as other objects present in BioMeld-Voxelyze simulation (e.g., vascular walls). Target morphologies are either generated by a simple generator producing simple 3D geometries (e.g., chain of voxel and cylinders) or by a more advanced generator based on compositional pattern producing networks.
- **BHM Simulator**. This component simulates BHM behavior in a specified environment for specified forces applied to BHM and specified physical side effects (temperature and gravity). The environment is defined by specifying other objects present in a simulation. During simulation, the component monitors the state of all objects and computes metrics related to the stability, flexibility and sensitivity of BHM and metrics reflecting pressure on environmental objects.

- **EA-based BHM Design Optimization Engine**. This engine searches for an optimal BHM design both in terms of morphology and material properties minimizing also the number of actuating voxels. The optimization engine will be based on an evolutionary/genetic algorithm that involves operators inspired by biological evolution, such as reproduction, mutation, recombination and selection. Candidate solutions to the optimization problem play the role of individuals in a population generated by Morphology Generator, while the optimization process is guided by metrics computed by BHM simulator. In each step of the evolutionary algorithm, evolutionary operators are applied to the current members of the population, resulting BMH designs are evaluated by the BHM simulator and the best designs are kept for the next evolutionary step. The evolutionary optimization algorithm stops either when the maximal number of evolutionary steps is reached or when the population contains sufficiently good BHM designs (expressed in terms of evaluation metrics computed by BHM simulator).
- **Utilities**. This sub-module encompasses several utility functionalities such as the camera view of the BHM model and model exporting functions.

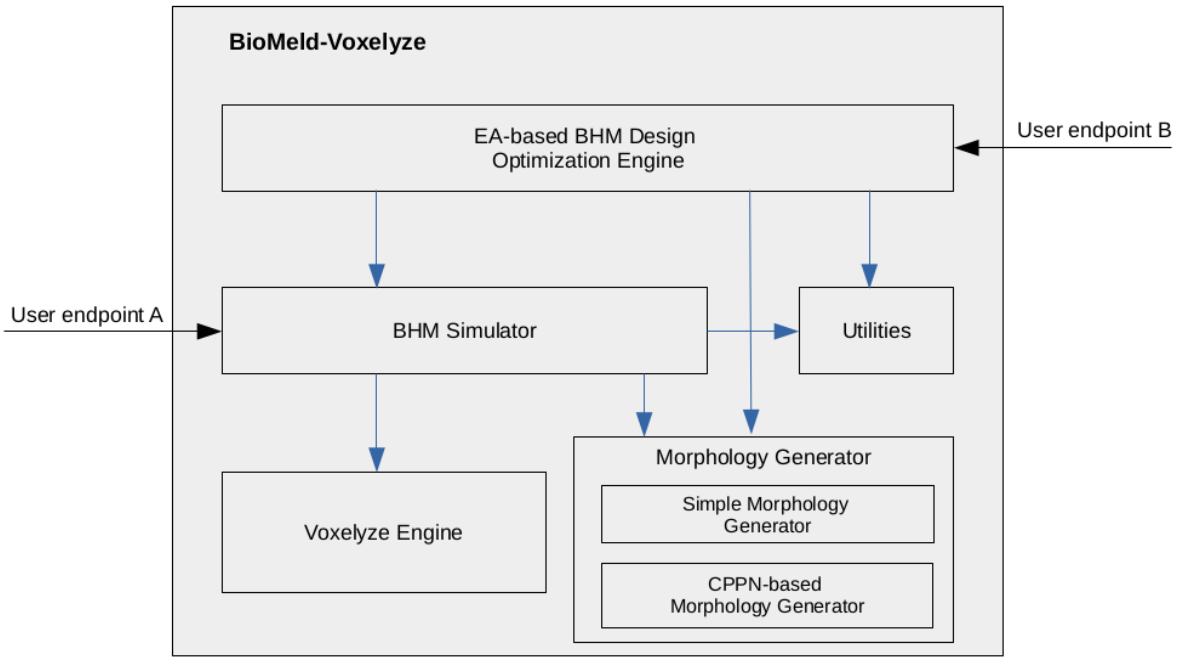

Figure 6. The architecture of the BioMeld-Voxelyze module. Blue arrows denote control-flow dependencies between different components of the module.

The architecture of the BioMed-Voxelyze module is shown in **Figure 6.** It can be seen that there are two user endpoints, which means that end users can run individual BHM simulations (user endpoint A) or can start BioMeld-Voxelyze to search for optimal BHM designs (user endpoint B). The BioMeld-Voxelyze simulator will be implemented in the C++ programming language in order to be directly

compatible with the Voxelyze library that is implemented in C++. The current version of Voxelyze[1] consists of 10 C++ classes:

1. CVX_Collision: class implementing collision detection at the level of voxels.
2. CVX_External: container for all external influences on a voxel such as forces and prescribed displacements.
3. CVX_LinearSolver: a linear solver for Voxelyze.
4. CVX_Link: class defining a solid link between two adjacent voxels and holding its current state.
5. CVX_Material: class defining the properties of a raw material with all information relevant to a physical material to be simulated.
6. CVX_MaterialLink: class defining the homogeneous material properties of a link connecting two voxels.
7. VX_MaterialVoxel: class defining a voxel type of a specific material (physical size, mass, moments of inertia).
8. CVX_MeshRender: Voxelyze mesh visualizer, the mesh can be drawn in an initialized OpenGL window by defining USE_OPEN_GL in the preprocessor and calling glDraw from within the drawing loop.
9. CVX_Voxel: class defining a specific instance of a voxel and holding its current state (voxel's physical characteristics, state, and links to other adjacent voxels.
10. CVoxelyze: the main class for simulating a configuration of voxels.

Physico-chemical simulation modules will be realized in the COMSOL Multiphysics software tool. COMSOL provides an integrated development environment and unified workflow for electrical, mechanical, fluid and chemical simulations. Its application builder can be used to develop independent custom domain-specific simulations, either by using graphical user interface or by scripting in its method editor.

The architecture of the module for **ML optimization of actuator parameters** is shown in **Figure 7**. This module will be implemented in Python, which is nowadays the default choice for machine learning (ML) and deep learning (DL) software development due to a large number of high quality ML/DL libraries and frameworks implemented in this language. We will utilize Tensorflow and/or PyTorch software frameworks as building blocks for our ML optimization of actuator parameters based on deep neural networks. The module for ML optimization of actuator parameters will also contain the following 4 components:

1. **MLP-NN Trainer**. This component performs the training of multilayer perceptron neural network (MLP-NN) models (SimulationDNN and ActuatorDNN) for a given training dataset and the specification of neural network architecture (the number of layers, the number of neurons per layer and activation functions) and learning hyperparameters (the number of epochs, batch size, loss function). From the architecture and hyperparameters specifications, this module builds MLP-NN in Tensforflow/PyTorch and uses existing loss function optimizers to determine MLP-NN weights and biases.

[1] https://github.com/jonhiller/Voxelyze

2. **MLP-NN Inference Engine.** This component makes predictions based on some previously trained MLP-NN model. Predictions are obtained according to the feed-forward mechanism described in Section 2.4, whose implementation is supported by Tensorflow/Pytorch.
3. **MLP-NN Evaluator.** This component evaluates a MLP-NN model by computing appropriate accuracy metrics (MAE, MSE, Pearson correlation and coefficient of determination) on data points from a given test dataset. MLP-NN Evaluator uses MLP-NN Inference Engine to obtain predictions from the model, and those predictions are compared to real (ground-truth) values from the test dataset when computing the previously mentioned metrics.
4. **Policy Reinforcement Learning**. This component trains PolicyDNN according to the previously trained SimulationDNN and ActuatorDNN in the reinforcement learning loop.

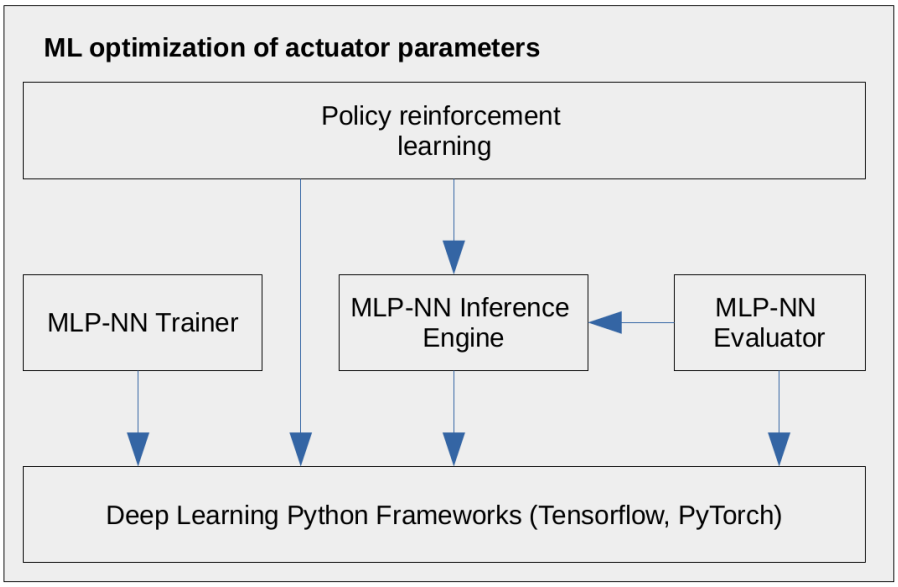

Figure 7. The architecture of the module for ML optimization of actuator parameters. Blue arrows denote control-flow dependencies between different components of the module.

3.3. DATA EXCHANGE FORMATS

Models created in the BioMeld-Voxelyze simulation module should be imported to COMSOL software (software in which physico-chemical modules will be realized). In the core Voxelyze library, models can be exported to the OBJ geometry definition file format. It is a simple data format that represents 3D geometry alone. It describes positions of vertices, positions of texture coordinate vertices, faces that each list of vertices defines, etc. Using the obj2stl library[2] it is possible to convert OBJ files to STL formats. This library is open source and written in Python. STL file format is native to stereolithography CAD software. This format is typically used for fast prototyping, 3D printing and CAD designs. The format itself supports both ASCII and binary representations, but obj2stl uses only ASCII specification. The algorithm for generating STL files can use triangulation or tessellation for creating faces of 3D objects. The generation of STL objects in obj2stl library is based on

[2] https://github.com/doitmaan/obj2stl

triangulation. COMSOL software is able to import and process STL file formats generated in this way. Only limitation of STL files is that it describes only the surface geometry of a 3D object without any information about its features. It does not contain information about material, texture, color, etc. This information can be imported to COMSOL in a separate parameter file. This parameter file is a textual file (extension .txt) where each line represents one important feature where one feature is described as *name expression/value description.* This parameter file is exported directly from the BioMeld-Voxelyze module.

4 VALIDATION AND VERIFICATION OF THE FRAMEWORK

4.1 BASIC CHARACTERIZATION OF THE MUSCLE-ACTUATOR:

Firstly, in order to evaluate the maturation of the muscle cells grown on the BHM, stainings of protein markers of muscle tissue such as MyHC and α-sarcomeric actinin will be performed, with these experiments the formation of aligned multinucleated myotubes will be evaluated. The contractile behavior of the 3D muscle actuator, e.i the force output, will be assessed under electrical stimulation by applying different voltages, ranging from 5 to 20V, at 1 Hz frequency of 2 ms. A force measurement platform, composed of a two-posts system made of 3D printed PDMS, will be used to determine the force contraction. Contractile tissues generate a force against the flexible posts which bend synchronized with the applied electrical pulse, to estimate the force exerted against the post, Eurler-Bernoulli's beam bending equation will be used. The force contraction of the system during long periods will be monitored through video recorded at different times, 0.25, 0.5,1, 3, 6 hours to test the muscle actuator lifetime.

4.2 CHARACTERIZATION OF THE BIOREACTOR

The design of the bioreactor and the force needed to deform it will be first simulated using Comsol Multiphysics. Candidate designs that can be bended in simulations using forces of a magnitude similar to the ones generated by the BHM will be produced by a combination of 3D printing and replica molding. Those candidates will be experimentally validated by measuring the bending produced as a response to the BHM actuation. Once the final design will be established, the number of functioning cycles without compromising the functionality will be experimentally assessed by performing cyclic bending of a magnitude 2 orders of magnitude larger than the exerted by the BHM.

4.3 TEST OF BHM (MUSCLE TISSUE + BIOREACTOR)

The 3D muscle tissue will be integrated in the bioreactor, and experiments similar to the previous section 4.1, will be performed to compare the force contraction and lifetime of the muscle actuator vs. the BHM. The best stimulation protocol will be applied to assess the motion of the BHM, including directionality, swimming speed and net displacement. Different software, such as Mathlab,

Python and Image J will be used to analyze the data. To improve the performance of the BHM, the integration of several 3D muscle actuators will be also evaluated.

5 FEEDBACK LOOP FOR BHM DESIGN

The control software will be developed as a standalone application using standard development libraries, enabling real-time data communication, storage, and visualization, and aimed to be solid, unified, and modular. A Bluetooth protocol will be employed for communication with the electronic module. The software will include calibration curve and, possibly, algorithms for further data processing. User-friendly data visualization and electronic module control will be ensured. The layout will be adapted according to the direct feedback provided by users during the employment of the system.

Basically, the software will allow:

- The set up of sensor bias (gate-to-source and drain-to-source voltage) and of stimulation conditions (voltage/current amplitude, frequency, waveform, duty cycle, stimulation duration). A part of/all these parameters may be reported in a separate page (expert mode);
- The setup of parameters in the front-end (e.g., signal amplification), only in an expert-mode configuration;
- The start/stop control for data transfer, and the data recovery in case of loss of communication (data will be stored in an integrated memory);
- Data visualization, saving and meta-function (e.g., zoom, change of x- and y-axis ranges).

Further functionalities will be introduced by interviewing the different groups involved in the testing activity of the BHM, by means of standard approaches of software engineering (e.g., Kanban and similar).

The graphic user interface (GUI) will be defined in coordination with all the group involved in the testing of the BHM.

The development will follow an incremental approach:

- Actual control on basic electronic configurations (sensor bias, stimulation, amplification) will be tested in laboratory with the employment of source meters, oscilloscopes, multimeters, impedance analyzers.
- The communication and data storage/retrieving will be assessed with mockup data, and by developing user cases to test the different functionalities;
- Data visualization, saving and recall will be assessed with mockup data (received by communication system), and so the meta-functions available at the GUI.

6 CONCLUSIONS

Bio-hybrid machines are systems that combine living and non-living components to perform specific tasks. They are designed to mimic the functionality of natural biological systems by integrating biological components, such as cells or tissues, with engineered materials or devices. These machines have the great potential to influence fields such as medicine, energy production, and environmental monitoring. However, our literature review showed that these machines are developed in individual research groups, and that knowledge for BHM design is usually based on intuition and on skills of involved researchers.

The main objective of the BioMeld project is to propose and develop a self-monitoring and self-controlling manufacturing pipeline of BHMs. This includes various aspects of BHM design including: choosing appropriate cells and materials for building devices, determining operating limitations (e.g. temperature, surrounding medium), expected life span of BHMs, verification & validation (V&V) procedures, control of selective and coordinated actuation, information processing within a device, etc.

The first task towards this common goal is the definition of a modeling and simulation framework that will guide the processes of design, quoting, manufacturing, verification, and reporting of BHMs. It is expected that this standardized design process will significantly reduce error-prone manual steps.

This deliverable presents the initial design of the proposed framework. All necessary methodologies are introduced and adequately explained. These includes: 3D voxel-based evolvable simulator (Voxelyze engine and encoding of morphologies using CPPN), estimation of observable parameters using complex networks approaches, physico-chemical modules for BHM behavior estimation (COMSOL Multiphysics software) and optimization of actuator parameters using AI approach with deep neural networks. Afterwards, the architecture of the framework is explained in detail, which represents the central part of this deliverable. The architecture is firstly described on a broad level, and after that all components are explained and discussed. Additionally, validation and verification activities and simulation feedback loop are briefly explained.

At the end, it is important to mention that this deliverable presents only the initial design and architecture of the modeling framework. The framework design will be improved and refined during the whole project (and reported at M18 and M36) to reflect obtained results.

7 REFERENCES

Cheney, N., MacCurdy, R., Clune, J., & Lipson, H. (2014). Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, *7*(1), 11-23.

Clune, J., & Lipson, H. (2011, August). Evolving three-dimensional objects with a generative encoding inspired by developmental biology (full article). In *ECAL 2011: The 11th European Conference on Artificial Life*. MIT Press.

Fortunato, S. & Newman, M. E. J. (2022). 20 years of network community detection, Nature Physics 18, 848–850, doi: 10.1038/s41567-022-01716-7

Goodfellow, I., Bengio, Y., & Courville, A (2016). Deep Learning. MIT Press.

Hagberg, A., Swart, P., & S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

Hiller, J. & Lipson, H. (2014). Dynamic Simulation of Soft Multimaterial 3D-Printed Objects. Soft Robotics. 1. 88-101. 10.1089/soro.2013.0010.

Hwangbo, J., Lee, J., Dosovitskiy, A., Tsounis, V., Koltun, V., & Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. Science Robotics 4(26), doi: 10.1126/scirobotics.aau5872.

Jin, D., Yu, Z., Jiao, P., Pan, S., Yu, P. S., & Zhang, W. (2023). A Survey of Community Detection Approaches: From Statistical Modeling to Deep Learning, IEEE Transactions on Knowledge and Data Engineering, 35(2): 1149-1170, doi: 10.1109/TKDE.2021.3104155

Liu, Y. Y., Slotine, J. J., & Barabási, A. L. (2013). Observability of complex systems. PNAS 110(7):2460-2465. doi: 10.1073/pnas.1215508110

Papavasileiou, E., Cornelis, J., & Jansen, B. (2021). A systematic literature review of the successors of "neuroevolution of augmenting topologies". *Evolutionary Computation*, *29*(1), 1-73.

Rossetti, G., Milli, L. & Cazabet, R. (2019). CDLIB: a Python library to extract, compare and evaluate communities from complex networks. Applied Network Science 4, 52, doi: 10.1007/s41109-019-0165-9

Savić, M., Ivanović, M., & Jain, L.C. (2019). Complex Networks in Software, Knowledge, and Social Systems, Springer Cham, doi: 10.1007/978-3-319-91196-0

Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz. P. (2015). Trust Region Policy Optimization. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:1889-1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M. & Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation, Proceedings of the International Conference on Learning Representations (ICLR).

Secretan, J., Beato, N., D Ambrosio, D. B., Rodriguez, A., Campbell, A., & Stanley, K. O. (2008, April). Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 1759-1768).

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, *8*, 131-162.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99-127.

Tarjan, R. E. (1972). Depth-first search and linear graph algorithms, SIAM Journal on Computing, 1 (2): 146–160, doi:10.1137/0201010

Tsompanas, M. A., Bull, L., Adamatzky, A., & Balaz, I. (2022). A Haploid-Diploid Evolutionary Algorithm Optimizing Nanoparticle Based Cancer Treatments. In *Cancer, Complexity, Computation* (pp. 237-251). Cham: Springer International Publishing.