Estimating the Learning Capacity of Bacterial Metabolic Networks

Bastien Mollet[1⋆], Paul Ahavi[2], Antoine Cornuéjols[1], Jean-Loup Faulon[2], Evelyne Lutton[1,3], and Alberto Tonda[1,3]

[1] EKINOCS, UMR 518 MIA-PS, INRAE, AgroParisTech, University of Paris-Saclay
[2] MICALIS Institute, INRAE, AgroParisTech, University of Paris-Saclay
[3] UAR 3611 ISC-PIF, CNRS, Paris, France

Abstract. Biocomputing has emerged as a promising field with the potential for energy-efficient computation, but standardized methods to evaluate living systems' problem-solving capabilities remain underdeveloped. To address this gap, we focus on bacterial systems and their metabolic adaptation. Using an *in-silico* model of bacterial behavior in varying environments, we propose a new framework for transforming supervised machine learning (ML) problems into a format solvable by biological systems. We then evaluate the framework's performance against other ML algorithms on standard classification and regression benchmarks. Experimental results show that bacterial metabolic networks often outperform linear methods and rival boosted trees, which are considered state-of-the-art for tabular data. A final ablation study suggests that the system's computational capacity may stem from its biological components rather than the translation tools used for the learning problem.

Keywords: Biocomputing · Leaning Capacity · Metabolic Networks.

1 Introduction

Living organisms can be viewed as information-processing systems that integrate input signals, perform computations, and produce results essential for survival. Since the 1930s and 1940s, the information processing metaphor has been used to understand biological functions across various scales, from molecular components to ecosystems [1,9,11]. Recently, there has been growing interest in harnessing the computing power of living organisms for practical applications in medicine, industry, and the environment [8,12,15]. Reviews highlight the potential of engineered bacteria for delivering tumor-specific drugs [3].

To assess the usability of living organisms for computations, their performance must be evaluated and quantified. This requires adopting an information-processing perspective, viewing organisms like bacteria as functions that associate inputs with outputs. Inputs can include chemical concentrations, temperature, and light, while outputs may involve changes in protein conformation, gene expression, cell growth, and motility.

⋆ Corresponding Author. Email: bastien.mollet@inrae.fr.

Researchers have turned to the machine-learning concept of "reservoir computing" (RC) to explore whether biological and physical systems can perform complex, time-dependent computations. In this approach, a system associates temporal inputs with outputs, performing a computation. To use a "reservoir" effectively, two key steps are required: (i) mapping the desired inputs to the system's actual inputs, and (ii) learning how to interpret the system's output. The goal is to explore the types of input-output relationships achievable through RC and their complexity. For example, studies ([7]) have asked if a "Liquid State Machine" (a type of reservoir) exists in Escherichia coli (*E. coli*). Other ([10] and [13]) explored the computational abilities of soft machines like plants used as reservoirs to compute complex time-dependent input-output relationships.

While previous work focuses on assessing how physical or biological systems can approximate specific target relationships within a reservoir computing framework, this paper takes inspiration from the Machine Learning framework. Specifically, we **evaluate the learning capacity of biological systems** using classical benchmark datasets for regression and classification tasks. We measure the complexity required for an in-silico learning algorithm to achieve the same predictive performance. For example, we compare the system's performance to that of linear methods (e.g., linear regression or perceptrons) or more sophisticated, non-linear methods, using ensembles of boosted trees like XGBoost [2]. This work expands on questions raised in [4], with technical differences in the framework and a deeper experimental analysis of the bacterium's contribution.

Our contribution in this paper is fourfold. (*i*) We propose to employ bacteria as part of supervised ML systems to perform regression and classification tasks; This entails performing back-propagation through the bacterium's surrogate model -here an Artificial Metabolic Network (AMN) [5]-, and we show how to do this (Section 2). (*ii*) We describe a new method to evaluate the capability of a learning algorithm (Section 3). (*iii*) We assess the learning capacity of a system combining a simple neural network to translate input data into a form readable by the AMN, with the AMN acting as a reservoir (Sections 3.1 and 3.4). (*iv*) Using an ablation/substitution approach, we evaluate the AMN's contribution to the problem-solving process and its potential for future computational devices (Section 4). In the conclusion section, we draw lessons from this new method for assessing the computational capacity of living organisms (indeed any learning algorithm), particularly its empirical results when applied to *E. coli*, and highlight future research directions.

2 Bio-computing with *E. coli*

Bio-computing aims to explore new substrates for computation by leveraging biological components, such as genes, that receive inputs (either endogenous or exogenous), transform those inputs, and produce outputs determined by their internal "rules." From this perspective, biological systems like *E. coli*, which take complex decisions based on environmental signals, can be viewed as signal-processing and computing units. This concept reveals the potential for using

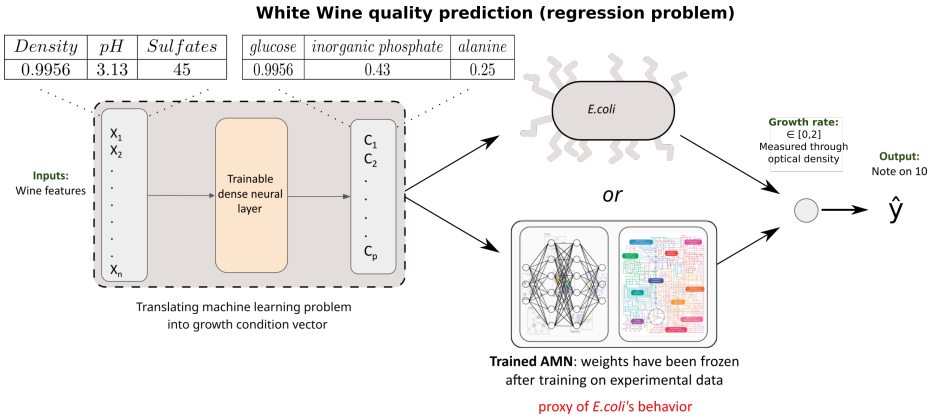

Fig. 1. Example of a bacteria used as a ML regressor. The values of input features (`Density`, `pH`, and `Sulfates`) are translated into concentrations of glucose, inorganic phosphate, and alanine, nutrients for the bacteria, using a trainable neural network. *E. coli*'s resulting growth rate is then interpreted as an output value for the regression problem. For full *in-silico* experiments, the bacteria can be replaced by an Artificial Metabolic Network modeling its behavior.

bacteria in computational tasks. As illustrated in Fig. 1, *E. coli* can be integrated into hybrid systems: an Artificial Neural Network (ANN) performs pre-processing, translating problem features into nutrients that serve as input signals for the bacterium. A second ANN is used for post-processing, converting the bacterium's output—such as growth rate or metabolite fluxes—into results suitable for tasks like regression or classification.

However, conducting experiments with live bacteria in *in vivo* settings can be impractical. To overcome this, we use an *in silico* surrogate: a genome-scale metabolic model (GEM) of *E. coli*, which makes it possible to simulate the bacterium's cellular metabolism for our computational experiments.

2.1 Using an *in-silico* surrogate of *E. coli*

In this study, we propose a novel hybrid approach that combines Machine Learning (ML) with Mechanistic Modeling (MM) to model the bacteria's metabolism. This leverages the complementary strengths of both methods. Mechanistic models are designed to explain biological phenomena by incorporating detailed physical and biochemical processes, but they face challenges in handling complex systems. Conversely, ML models can accurately predict complex biological outcomes even without a mechanistic understanding; however, they require large, representative datasets for training. This complementarity has led to hybrid approaches, such as the Physics-Informed Neural Network (PINN) framework [14]. One hybrid model resulting from these considerations is the AMN model recently presented in [5]. One key property of this hybrid model being that *it makes it possible to backpropagate errors through the whole system*, which is crucial to be able to train the pre-processing ANN that translates the problem features into

a signal for *E. coli*. Therefore, for the rest of this paper, we use the AMN as trained and described in [5] in place of *E. coli*, as in Fig. 1. For transparency and reproducibility, the code is available on GitHub[1]

2.2 An *E. coli* based processing pipeline for ML tasks

In order to use *E.coli* – replaced for *in-silico* experiments by an AMN, as described in section 2.1 – we designed a processing pipeline (Fig. 1) that associates an input for the ML task with an output used for prediction. Fig. 1 illustrates this for the problem of predicting the rating of a wine described by a set of features $X_1, X_2, \ldots, X_n$. Since bacteria react to a set of average concentrations $C_1, C_2, \ldots, C_p$, we need to translate the features describing an example into a set of average concentrations that the bacteria can perceive and react to. In our approach, this is done in three steps: (*i*) First, the input data are normalized using the MinMaxScaler function of the sklearn Python library on the training set. (*ii*) Next, if the input space dimensionality exceeds that of the bacteria ($n > p$), PCA is applied to retain the first p principal components; otherwise, the inputs are used as is in the next step. (*iii*) Finally, a single-hidden-layer ANN is trained to translate the problem inputs into inputs for the bacteria in such a way that the training error is minimized. In this study, we use a hidden layer with 200 neurons and a Relu activation function. The principal trainable part of the overall system is therefore located *before* the AMN, whereas in the classical RC framework, the trainable part is located *after* the reservoir. In this framework, a final trainable layer converts the output signal i.e. growth rate into the predicted output $\hat{y}$, this layer has only one node. Its has no activation function for regression tasks (corresponding to a rescaling) and for classification task the activation function was sigmoid, sharpened with a factor $10e4$. This is possible because the output error can be backpropagated through the AMN.

3 Learning capacity of the *E. coli* pipeline

3.1 Dataset selection

The datasets used in our experiments are taken from the OpenML [17] benchmark suites OpenML-CC18 (classification) and OpenML-CTR23 (regression) [6]. Starting from the full suites (72 classification and 40 regression problems), we further selected datasets containing only real-valued features, and for classification we considered only binary classification tasks, with a final selection of 24 regression and 20 classification datasets. To illustrate possible device behaviors, from the datasets that met the conditions, three were selected for classification on the one hand, and for regression on the other. They were chosen on the basis of the size of the dataset, neither too small to be too easy, nor too large to be too

[1] https://github.com/bmollet01/LCBMN.

costly in terms of time complexity, and on the basis of preliminary results to exclude learning problems that were too easy or those requiring multiple outputs. More specifically, the `energy_efficiency` and `phoneme` datasets were selected to be challenging for the approach.

3.2 Methodology

The approach is as follows: for each ML task, we measure the predictive performance of the full pipeline after training (details below) and then attempt to achieve *same average performance* using either a linear model or XGBoost, with parameters such as the number and depth of trees controlled (see Fig. 2). To simplify computational capacity evaluation, we use single-node trees ("decision stumps") in XGBoost, adjusting the number of trees to match the pipeline's performance. Section 4 further discusses methods for evaluating the respective roles of the translation component and of the bacterium.

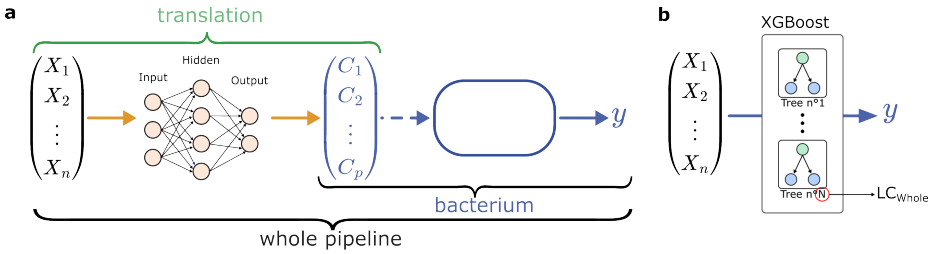

Fig. 2. (**a**) The proposed pipeline is composed of a first stage with a trainable NN, translating the problem's feature values into concentrations of nutrients for the bacteria, whose growth rate is then interpreted as the output. (**b**) XGBoost can be used to estimate the learning capacity of another ML method, increasing its number of trees until a target performance is reached.

3.3 Capacity measurement

In order to measure the learning capacity of a given system on a given task, we first measure its predictive performance on that task using a 5-fold cross validation. We then iteratively apply XGBoost increasing the number N of single-node trees until it reaches the same predictive performance, again using a 5-fold cross validation.

In this section, we focus on the learning capacity of the whole pipeline, thus considering *LC–whole* (Fig. 2 (b)). As shown in Fig. 3, where the x-axis stands for the number of node trees (here in log scale) and the y-axis stands for the error rate (for classification tasks) or Q^2 score (for regression tasks). It appears from Fig. 3 that the number N_{est} of XGBoost stumps (in a log scale), tends to plateau as the number of decision or regression trees increases.

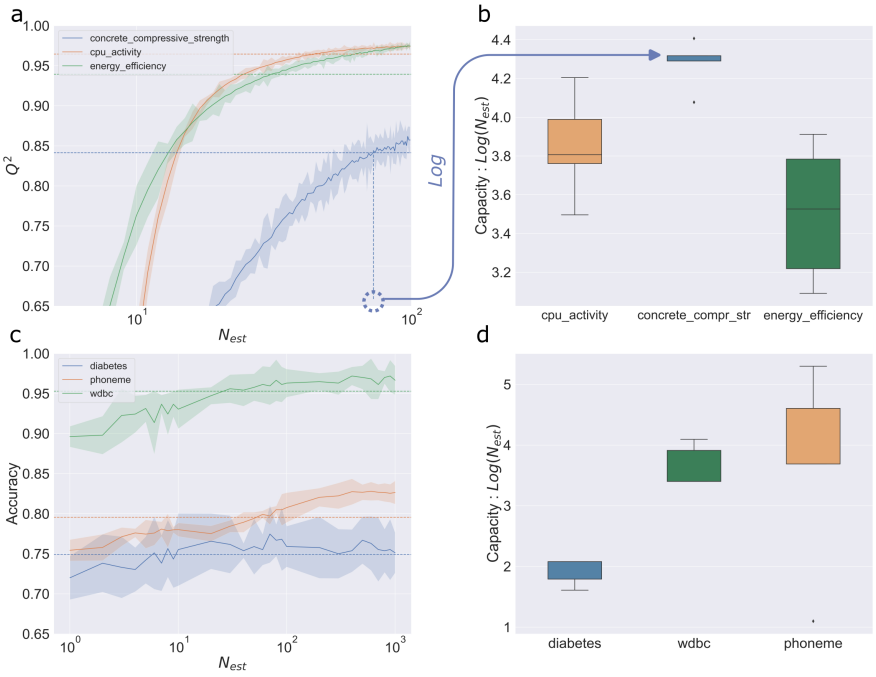

Fig. 3. Evaluation of the learning capacity of the pipeline with *E. coli* on three regression tasks (a & b) and on three classification tasks (c & d).

In Fig. 3, (a) and (c), the *dotted horizontal lines* indicate the performance of the pipeline for each learning tasks. Thus, the number N_{est} of XGBoost stumps (in a log scale), required to reach a given level of performance can be read on the x-axis. The learning capacity of the pipeline is displayed in (b) for all repetitions and for the three *regression tasks* and in (d) for the classification tasks. For instance, in (a), $\sim$ 75 stumps are needed in order to attain the performance of the pipeline on the `concrete_compressive_strength` task (i.e. $Q^2 = 0.84 \pm 9.4e{-}5$), corresponding to a capacity of $\log 75 \approx 4.3$ reported in (b) with standard deviation.

Looking at *classification tasks*, we observe three interesting behaviors. In one, `wdbc`, both the pipeline and XGBoost achieve high accuracy (≥ 0.97), and XGBoost needs a large number of decision stumps to reach this level. The pipeline's learning capacity is therefore high. For the `diabetes` dataset, we see a completely different case, where the classification problem seems difficult for both the pipeline and XGBoost, with an accuracy that remains around 0.75 for both and whatever the number of decision stumps in XGBoost between ≈ 10 and 10^3. The pipeline capacity is therefore imprecise, on the order of $[\log 10, \log 1000]$. We have reported the lowest value here in Fig. 3 (d). Finally, there is an intermediate type of problem exemplified by the dataset `phoneme`, where the problem is hard and XGBoost needs a high number of decision stumps to achieve pipeline performance, so the pipeline capacity is high. This underlines the value of examining both learning performance and capacity, as they provide complementary information.

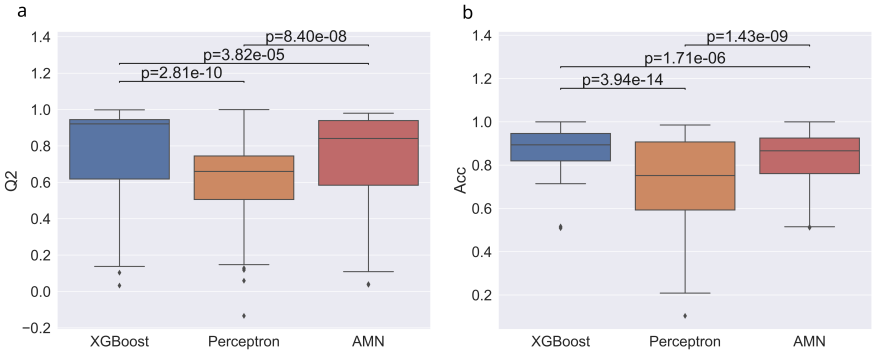

Fig. 4. Method comparison between AMNs, linear perceptron and XGBoost using Wilcoxon Signed rank test. (**a**) Q^2 scores of different methods on 10 different regression problems with $n = 5$ replicates. Here the Q^2 score corresponds to R^2 performed on test data so it is a metric of models' predictive power.(**b**) Accuracy scores of AMNs, linear perceptron and XGBoost classifier on 20 classification problems with $n = 5$ replicates.

3.4 Method comparison

Fig. 4 sums up the results for 10 datasets of the *regression tasks* (a), and for 20 datasets for classification tasks (b). On the regression tasks (a), the pipeline with *E. coli* reaches a performance in between the one of linear regression and using XGBoost with 50 regression trees of depth 30. This setup for XGBoost has been set through hyperparameter optimization to maximize average performance on all datasets. The methods were compared using a Wilcoxon signed-rand text with two-sided hypothesis and displaying the *p-value*.

For some specific datasets such as `energy_efficiency` (a regression task), the pipeline has a lower performance than linear regression. However, this is not surprising since it is known that this dataset is well described by linear relations between only 3 of its 8 features.

On the 20 datasets for *classification* (Fig. 4 (b)), the same trend is observed, with the pipeline having a performance that lies between linear classification and XGBoost. From this analysis, we can tentatively conclude that the *E. coli* based pipeline **outperforms linear methods for nonlinear problems**, and **underperforms a high-performance nonlinear method like XGBoost** given enough trees.

4 Evaluating the contribution of the bacterium

As illustrated in Fig. 2, the bacterium is but one element in the computational pipeline. A critically important question is to evaluate its contribution to the learning process. We do this in two ways.

First, we *evaluate what is required of the translation step*, performed by a trainable one hidden layer ANN, in order to solve the learning problem when the bacterium is replaced by a simple and fixed *linear function* $\mathbb{R}^p \to \mathbb{R}$ (Fig. 5

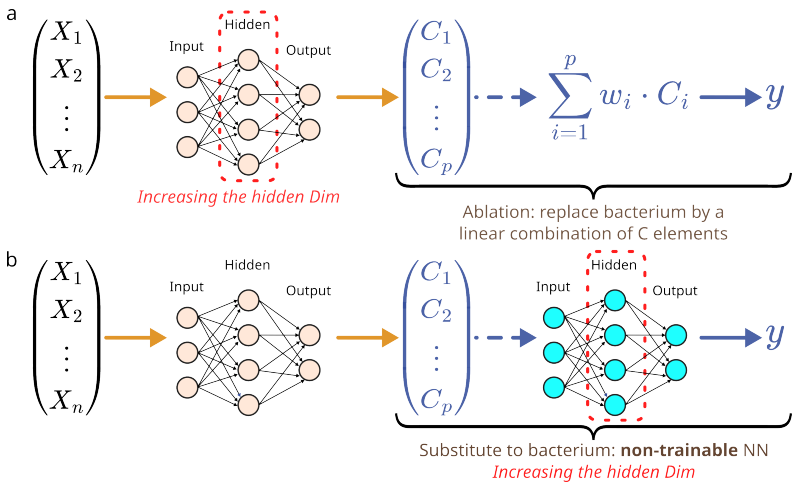

Fig. 5. (**a**) Architecture used for the **ablation study** where the bacterium is replaced with a simple linear transformation. Note that the trainable ANN has the same number of trainable weights than the translation ANN of the whole pipeline. (**b**) Architecture used for the **substitution study** where the bacterium is replaced by a one hidden layer ANN with frozen randomly initialized weights and 200 neurons in the hidden layer of the translation ANN (see Section 4).

(a)). We call this the **ablation study**. The idea is that, the larger the number of hidden neurons required to solve the task on the translation side, the higher is the contribution of the bacterium since this shows that it cannot easily be replaced with just a linear function.

Second, we set the size of the translation ANN to 200, as this gives a performance close to the maximum achievable value (Fig. 6). While this ANN is still learnt for optimizing the performance on each problem, we now *use a* ***frozen*** *one hidden layer ANN as a substitute for the bacterium* so that it can implement non linear functions (Fig. 5 (b)). We call this the **substitution study**. This time, we *vary the size of the hidden layer* of the substitute so as to evaluate its required capacity for the learning problem to be solved. It is assumed that the larger this size, the bigger is the role of the bacterium.

It is expected that (*i*) increasing the number of hidden neurons in the bacterial substitute will lead to an increase in performance, which could reach that of the entire pipeline. If so, this would give us an estimate of the bacterium's capacity itself (i.e. excluding the translation layer) and shows that it does the work of a one hidden layer ANN. It is also possible that (*ii*) even by increasing the number of hidden neurons, the performance of the whole pipeline is never matched, in which case we could hypothesize that bacteria have a higher capacity than a single-hidden-neuron ANN using Relu functions. A third case is (*iii*) where the neural substitute enables the system to outperform the entire pipeline. In this case, the bacterium can be said to have a lower capacity than that of a single hidden layer ANN.

In both the ablation and substitution studies, the trainable part lies in the translation component, while the bacterium is replaced either by a fixed linear function (ablation) or by a fixed one hidden layer ANN with the Relu activation function (substitution). $n = 20$ replicates were randomly generated for each substitute size. For each replicate, weights were drawn from a uniform distribution in $\left[-\sqrt{\frac{6}{p_{in}+p_{out}}}, \sqrt{\frac{6}{p_{in}+p_{out}}}\right]$, with p_{in} the number of inputs and p_{out} the number of outputs (default initialisation of the keras python library). For the replicates of the linear function, the weights were drawn from a uniform distribution in $[-1, 1]$.

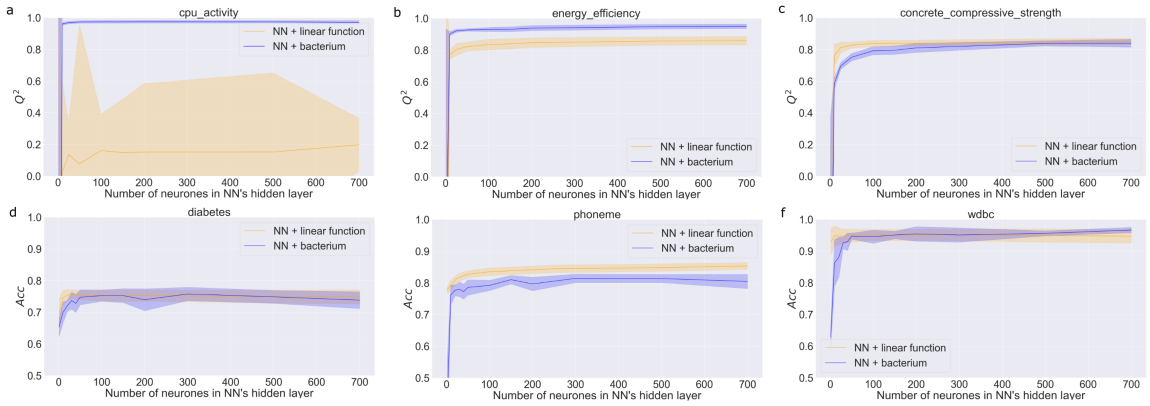

Fig. 6. Ablation study on three regression tasks (a-c) and three classification tasks (d-f). The performances of our pipeline (blue lines) with increasing number of neurons in the translation hidden layer (ranging from 1 to 700) were compared to the performances of the pipeline where *the bacterium is replaced by a simple linear transformation*, as presented in Fig. 5(a). Mean and variance were obtained with $n = 20$ random initializations of the linear layer's weights.

4.1 Results of the ablation study

The question is: does the learning capacity can be explained entirely by the translation component?

For *regression*, we base our observations on the three tasks reported in Fig. 6 (a, b, c). In all cases, it appears that either it is impossible for the translation component to overcome the absence of the bacterium for all numbers of hidden neurons, leading to impoverished performance (as is apparent for the `energy_efficiency` and the `cpu_activity` datasets), or it can equal the whole pipeline performance as for the `concrete_compressive_strength` dataset.

For *classification* tasks, the results in Fig. 6 (d, e, f) on three datasets show a somewhat different picture. Here, the translation component can manage to learn as well as or better than the whole pipeline.

Thus, regarding the `cpu_activity` dataset, for regression, the pipeline performs much better than a linear model, and the ablation study shows that this cannot be explained by the performance of the translation component. Conversely, on the `diabetes` dataset related to a classification task, the good performance of the pipeline could be due to the translation component.

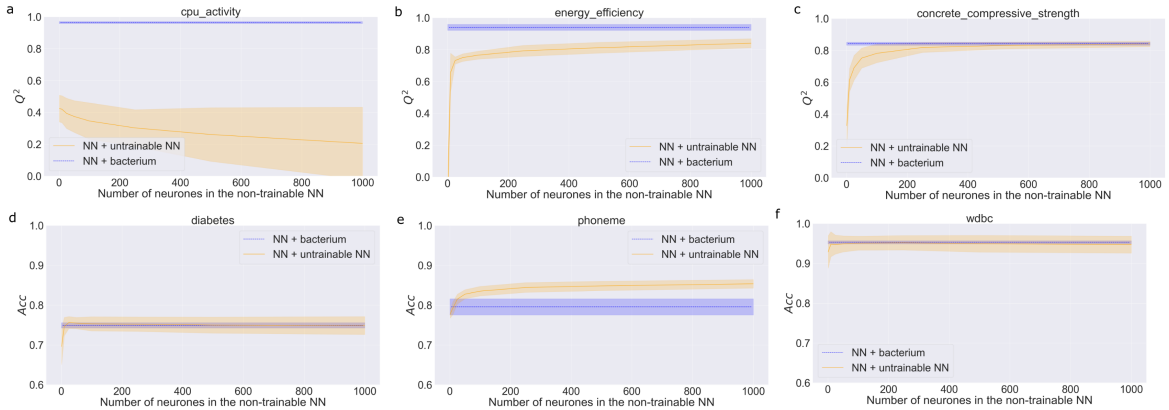

Fig. 7. Substitution study on three regression tasks (a-c) and three classification tasks (d-f). The performances of the whole pipeline (Fig. 4), here represented as dashed blue lines, were compared to the performances of the framework presented in Fig. 5(b) where *the bacterium is replaced by a one hidden layer ANN*. The mean and variance, were obtained with $n = 20$ random initializations of the non-trainable ANN with its hidden layer dimension ranging in $[1 - 1000]$.

4.2 Results of the substitution study

Here, the question is: what is the level of non linearity and expressivness of the bacterium? This being measured by the number of neurons in a one hidden layer neural network that reaches the same level of perfomance if possible. (Fig. 5 (b)).

The experimental results for the substitution study (Fig. 7) confirm the difference between regression problems and classification ones found in the ablation study.

On *regression tasks*, the bacterium based pipeline exhibits greater or equal performance than the substitute based bacterium, for all number of hidden neurons that were tested. For instance, the examination of the `cpu_activity` dataset shows that the performance of the bacterium-based pipeline could not be matched by any of the one hidden layer ANN with Relu activation functions tested. This suggests that the bacterium performs a computation that requires more than what a one hidden layer ANN can compute. This analysis also applies, to a lesser extent, to the case of `energy_efficiency` dataset.

The situation is reversed for *classification tasks*. On the `wdbc` dataset, the performance of both the bacterium-based pipeline and of the substitute based pipeline are very good, This is true whatever the number of hidden neurons in the translation component (ablation study) or in the substitute based pipeline (substitution study). It may be hypothesized, also using Fig. 4, that a simple model captures all there is to capture in this dataset and that more powerful models are not necessary.

In the case of the `phoneme` dataset, the performance of the pipeline with substitute bacteria is superior to that of the pipeline with bacteria. This is not surprising, since it was already the case with a simple linear transformation instead of the bacteria. What is more surprising is that performance does not increase much with the increasing number of hidden neurons in the substitute bacteria. This suggests that for this dataset, the translation component does all the learning whatever the component after it, as if, in a deep NN, the first layers were enough to learn the task whatever the weights of the upper layers were. Except that here, the first layers cannot overcome the bias introduced by the upper layers, i.e. the bacterium. This is another testimony that the bacterium acts as a bias which, as any bias does, can help (the case of the `cpu_activity` dataset) or hinder learning (the case of the `phoneme` dataset).

These findings are consistent with state of the art knowledge about RC. In their 2019 review, Tanaka et al. [16] stated that (*i*) preprocessing is a crucial step for RC, and (*ii*) that RC performances are subject to hyperparameters optimization. We did not conduct any optimization here since we took the bacterium in one specific condition, but actually bacteria offers a wide variety of optimizable variables ranging from environmental conditions to genetic modifications. It is therefore entirely conceivable to seek to optimize the functioning of a bacterium according to the problem under study.

5 Conclusions

Given the energetic efficiency of bacteria in transforming input signals into output, and therefore in performing calculations, and the wealth of applications, particularly in the healthcare field, where they could be used, it is important to assess the extent to which they can help in learning tasks. For example, in the detection of a subject's pathology using a probe applied directly to the skin or to a blood sample.

In this paper, to answer the general question of whether the metabolism of living organisms such as bacteria can be used for data processing, we presented a **method** based on Machine Learning evaluation protocols. We proposed to assess a system's learning capacity by considering a parameterizable model that can equal its performance. Here, we have considered the XGBoost method, with the number of trees (for regression) or the number of decision stumps (for classification) as the hyper-parameter. We thus evaluated the learning capacity of a pipeline using a bacterial metabolic network for *E. coli*. A series of experiments with datasets corresponding to regression and classification tasks led to

the conclusion that using *E. coli* as a computational device provides *a learning capability between linear methods and the XGBoost* algorithm known for its high level of performance for non-linear problems.

In order to evaluate the contribution of the bacteria to the learning task, we carried out an *ablation* study, replacing it with the simplest transformation between the output of the bacteria and the output corresponding to the learning task, i.e. a linear function. At the same time, we also introduced the principle of a *substitution* study to assess the degree of non-linearity and expressiveness of the organism in the learning pipeline. We found that, for regression tasks, E.coli behaved as a positive bias and exhibited a high degree of expressivity, whereas, for classification tasks, this was less so.

This study brings a first characterization of the data processing potential of *E. coli*. Further work, currently in progress, includes (*i*) building a broader overview of the types of learning problems that can benefit from the use of bacteria in a learning pipeline, (*ii*) replicating *in vivo* the experiments detailed in this article and (*iii*) carrying out experiments in which multiple outputs from bacteria are taken into account.

References

1. Balasubramaniam, S., Somathilaka, S., Sun, S., Ratwatte, A., Pierobon, M.: Realizing molecular machine learning through communications for biological ai. IEEE Nanotechnology Magazine (2023)
2. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. ACM (8 2016). https://doi.org/10.1145/2939672.2939785
3. Chowdhury, S., Castro, S., Coker, C., Hinchliffe, T.E., Arpaia, N., Danino, T.: Programmable bacteria induce durable tumor regression and systemic antitumor immunity. Nature Medicine **25**(7), 1057–1063 (Jul 2019). https://doi.org/10.1038/s41591-019-0498-z
4. Faulon, J.L., Ahavi, P., Hoang, T.N.A.: Reservoir computing with bacteria. bioRxiv (2024). https://doi.org/10.1101/2024.09.12.612674
5. Faure, L., Mollet, B., Liebermeister, W., Faulon, J.L.: A neural-mechanistic hybrid approach improving the predictive power of genome-scale metabolic models. Nat Commun **14**(1), 4669 (Aug 2023). https://doi.org/10.1038/s41467-023-40380-0
6. Fischer, S.F., Feurer, M., Bischl, B.: OpenML-CTR23 – A curated tabular regression benchmarking suite. In: AutoML Conference 2023 (Workshop) (Aug 2023)
7. Jones, B., Stekel, D., Rowe, J., Fernando, C.: Is there a liquid state machine in the bacterium escherichia coli? In: 2007 IEEE Symposium on Artificial Life. IEEE (Apr 2007). https://doi.org/10.1109/alife.2007.367795
8. Kieffer, C., Genot, A.J., Rondelez, Y., Gines, G.: Molecular computation for molecular classification. Advanced Biology **7**(3), 2200203 (2023)
9. Monod, J.: On chance and necessity. Springer (1974)
10. Nakajima, K., Hauser, H., Li, T., Pfeifer, R.: Information processing via physical soft body. Scientific reports **5**(1), 10487 (2015)

11. Nicholson, D.J.: Is the cell really a machine? Journal of theoretical biology **477**, 108–126 (2019)
12. Pandi, A., Koch, M., Voyvodic, P.L., Soudier, P., Bonnet, J., Kushwaha, M., Faulon, J.L.: Metabolic perceptrons for neural computing in biological systems. Nature communications **10**(1), 3880 (2019)
13. Pieters, O., De Swaef, T., Stock, M., Wyffels, F.: Leveraging plant physiological dynamics using physical reservoir computing. Scientific Reports **12**(1), 12594 (2022)
14. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics **378**, 686–707 (2019)
15. Rizik, L., Danial, L., Habib, M., Weiss, R., Daniel, R.: Synthetic neuromorphic computing in living cells. Nature communications **13**(1), 5602 (2022)
16. Tanaka, G., Yamane, T., Héroux, J.B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., Hirose, A.: Recent advances in physical reservoir computing: A review. Neural Networks **115**, 100–123 (2019)
17. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning (Jul 2014). https://doi.org/10.1145/2641190.2641198