Hybrid models enabling neural computations with metabolic networks

Léon Faure[1,2], Bastien Mollet[2,3], Wolfram Liebermeister[1,4], and Jean-Loup Faulon[1,2,5*]

[1]University of Paris-Saclay, Saclay, France, [2]MICALIS Institute, INRAe, Jouy-en-Josas, France, [3]ENS Lyon, Lyon, France, [4]MaIAGE, INRAe, Jouy-en-Josas, France, [5]Manchester Institute of Biotechnology, University of Manchester, Manchester, UK.
*Corresponding author: Jean-loup.Faulon@inrae.fr, ORCID 0000-0003-4274-2953

Abstract

Constraint-based mechanistic models have largely been exploited to predict the phenotype of microorganisms in different environments. However, phenotype predictions are limited in quality unless labor intensive experiments including the measurement of media uptake fluxes, are performed. We show how hybrid - mechanistic and neural – models provide ways to improve phenotype predictions. Our hybrid models named Artificial Metabolic Networks (AMNs) surrogate constraint-based modeling, make metabolic networks suitable for backpropagation and, consequently, can serve as an architecture for machine learning. We first show how learning principles brought by AMNs can replace the optimization principle of constraint-based modeling with excellent performances for various *in silico* training sets. We then illustrate how AMNs outperform mechanistic models with *Escherichia coli* growth rates measured in 110 different media compositions reaching regression coefficients > 0.76 on cross-validation data. We expect our hybrid AMN models to enhance constraint-based modeling and to prompt new biotechnological applications.

Keywords: Artificial Neural Network, Metabolic Network, Mechanistic Modeling, Flux Balance Analysis, Scientific Machine Learning, Hybrid Modeling.

Abbreviations: **AMN**: Artificial Metabolic Network, **ANN**: Artificial Neural Network, **FBA**: Flux Balance Analysis, **GD**: Gradient Descent, **LP**(QP): Linear (Quadratic) Programming, **ML**: Machine Learning, **MM**: Mechanistic Modeling, **PINN**: Physics Informed Neural Network, **RNN**: Recurrent Neural Network. $\mathbf{R^2}$: Regression coefficient calculated on training set. $\mathbf{Q^2}$: regression coefficient calculated on cross-validation sets or independent test sets (not seen during training).

Introduction

The increasing amounts of data available for biological research bring the challenge of data integration with Machine Learning (ML) to accelerate the discovery process. The most compelling achievement within this grand challenge is protein folding recently cracked by AlphaFold[1], which in the last CASP14 competition predicted structures with a precision similar to structures determined experimentally. Following this foot step, one may wonder if in the future we will be able to use ML to accurately model whole cell behaviors.

The curse of dimensionality[2], *i.e.*, the fact that fitting many parameters may require prohibitively large data sets, is perhaps the biggest hurdle that prevents using ML to build cell models. Obviously, cells are far more complex than single proteins and since the amount of data needed for ML training grows exponentially with the dimensionality[2], as of today, ML methods have not been used alone to model cellular dynamics.

For the past decades mechanistic (mathematical) models (MMs) have been developed to simulate whole-cell dynamics (*cf*. Thornburg *et al.*[3] for one of the latest models). These models encompass metabolism, signal transduction, as well as gene and RNA regulation and expression. Cellular dynamics being tremendously complex, MMs are generally based on strong assumptions and oversimplifications and ultimately suffer from capacities of making predictions beyond the assumptions and the data used to build them.

The MM and ML approaches are based on two contrary paradigms. While the former is aimed at understanding biological phenomena with physical and biochemical detail, it has difficulties handling complex systems; the latter can accurately predict the outcomes of complex biological processes even without an understanding of the underlying mechanisms, but require large training sets. The pros of one are the cons of the other, suggesting that hybrid approaches should be developed. In particular, MMs may be used to tackle the dimensionality curse of ML methods. For instance, one can use MMs to produce *in silico* data which can be added to experimental data increasing the training set sizes for ML. However, with that strategy, if the model is inaccurate, ML will be trained on erroneous data. One can also embed MMs within the ML process, in this strategy ML and MMs are trained together and the model parameters can be estimated through training, increasing the model predictive capacities. The issue with this strategy is the difficulty of making MMs amenable to training.

In the current paper we propose an MM-ML hybrid approach in which a whole-cell constraint-based model (CBM), Flux Balance Analysis (FBA), is embedded with ML. For the past three decades, FBA has been the main approach to study the relationship between nutrient uptake and the metabolic phenotype (*i.e.*, the metabolic fluxes distribution) of a given organism, *e.g.*, *E. coli*, with a model iteratively refined over the years[4]. FBA assumes that the metabolic phenotype is at steady state, *i.e.*, a phenotype that is constant in time and in which all compounds are mass-balanced. Usually such a steady state is assumed to be reached in the mid-exponential growth phase. The search for a steady state happens in the space of possible solutions that satisfies the constraints of the metabolic model, *i.e.*, the mass-balance constraints according to the stoichiometric matrix as well as upper and lower bounds for each flux in the distribution. In addition, FBA employs an optimality principle, with one principal objective (usually the 'biomass' production flux) and possibly secondary objectives (*e.g.,* minimize the sum of fluxes in parsimonious Flux Balance Analysis (pFBA), or the flux of a metabolite of interest). For example, to predict a growth rate in a given environment, one would designate the 'biomass' flux (describing the growth rate) as the objective to maximize and set up non-zero upper bounds on some uptake fluxes. These bounds depend on the amounts of molecule transporters, which may change between conditions depending on the cell metabolic strategy. Therefore, finding realistic, condition-dependent bounds on the uptake fluxes requires labor-intensive measurements. In more sophisticated CBM approaches such as molecular crowding FBA[5] (mcFBA) or Resource Balance Analysis[6] (RBA), constraints on the resource availability and allocation are added to obtain more biologically plausible metabolic phenotypes, but parameterizing such models requires additional data. To set bounds and validate prediction results, fluxomic data can be obtained from experiments. With isotopic labeling (like 13-C), one can follow the path of a nutrient in the metabolic network[7]. Based on metabolomics, one can derive metabolic fluxes from metabolite concentrations and possibly in a time-resolved approach[8]. With transcriptomics, RNA sequencing data is used as input for models to estimate fluxes indirectly, even at the single-cell level[9]. Data obtained from several -omics methods can be integrated, constituting a state-of-the-art multi-omics data integration for flux prediction[10–12], which usually makes use of FBA alongside ML methods. Indeed, ML approaches were naturally developed to efficiently integrate such data and to enhance predictions by FBA. However, as described by Sahu *et al*.[13], the interplay between FBA and ML still shows a gap: some approaches use ML results as input for FBA, others use FBA results as input for ML, but none of them embed FBA into ML, as we do in this paper with the Artificial Metabolic Networks (AMNs) hybrid models.

Hybrid models have recently been developed under different names in biology for signaling pathways and gene-regulatory networks (Knowledge Primed Neural Network[14], Biologically-Informed Neural Networks[15]) and in physics where models solving partial differential equations (PDEs), such as Physics

Informed Neural Network[16] (PINN), are available in open source repositories such as SciML.ai[17]. The goal of these emerging hybrid modeling methods is to generate models that comply well with observations or experimental results via ML, but that also use mechanistic insights from MM. The advantages of hybrid models are two-fold: they can be used to parametrize MM methods through direct training and therefore increasing MM predictability, and they enable ML methods to overcome the dimensionality curse by being trained on smaller datasets because of the constraints brought by MM.

The AMN models shown here fit in the emerging hybrid modeling field. AMNs bridge the gap between ML and FBA by computing steady-state metabolic phenotypes with different methods, that can be embedded with ML: a method based on recurrent neural network (RNN), and two trainable methods solving respectively linear programs (LP) and quadratic programs (QP). All these methods are relying on custom loss functions surrogating the FBA constraints. By doing so, our AMNs are mechanistic models, determined by the stoichiometry and other FBA constraints, and also ML models, as they are used as a learning architecture.

We showcase our AMNs with a critical limitation of classical FBA for experimentalists. As already discussed, realistic and condition-dependent bounds on medium uptake fluxes are critical for growth rate (or other fluxes) predictions, but there is no simple conversion from extracellular concentrations, *i.e.*, the controlled experimental setting, to such bounds on uptake fluxes. In fact, these bounds depend on the internal allocation of protein resources to transporters, which itself depends on the cell's metabolic state - the state we are trying to predict. Some methods, such as satFBA[18], assumes fixed transporter levels and converts medium concentrations to possible uptake fluxes by kinetic models, relying on a Michaelis-Menten value for each uptake reaction. Consequently, this MM approach relies on more assumptions and requires more data than a classical FBA, and does not integrate any large-scale regulation or resource allocation phenomena. More advanced and accurate CBM methods such as Resource Balance Analysis (RBA) are more difficult to set-up and harder to parametrize. As a result, many users rely on classical FBA, which cannot reliably predict growth rates from media compositions. AMNs are used in this study for tackling the same issue: predicting metabolites uptake fluxes from their external concentrations. To do so, where satFBA uses transporter kinetics with parameters that need to be acquired through additional experimental measurements, AMNs use a pre-processing neural layer that is accessible for learning. This neural pre-processing layer aims to capture, effectively, all effects of transporter kinetics and resource allocation in a particular experimental setting, predicting the adequate input for a metabolic model to give the most accurate steady-state phenotype prediction possible. Consequently, AMNs provide a new paradigm for the

prediction of fluxes and growth rates: instead of relying on a constrained optimization principle performed for each condition (as in classical FBA), we use a learning procedure on a set of example flux distributions that attempts to generalize the best model for accurately predicting the metabolic phenotype of an organism in different conditions.

Results

As an introductory work, we show how the aforementioned idea of knowledge-primed neural networks inspired an in-house method based on branching ratios of metabolites going to different reactions (Figure 1). This preliminary work having limitations, we then showcase more versatile methods that surrogate FBA with gradient-backpropagation compatibility: a method for solving the linear program (LP) of FBA, and a quadratic program (QP) method solving FBA when measured fluxes (like growth rate) are provided. These two solvers have adequate gradient and loss functions satisfying the constraints of an FBA framework (Figure 2). Then, we used these methods inside hybrid models that can directly learn from sets of flux distributions (Figure 3). These flux distributions used as learning references (*i.e.*, the training sets) are either acquired experimentally or produced through FBA simulations (Table 1 and Figure 4). Finally, we developed a non-trainable AMN reservoir to showcase how the predictive power of FBA can be improved. Indeed, once the AMN has been trained on adequate *in silico* FBA data, we can fix its parameters, resulting in a gradient backpropagation compatible reservoir that mimics FBA (Figure 5). This reservoir can then be used to tackle the above-mentioned issue of unknown uptake fluxes: adding a pre-processing neural layer and training this layer with an experimental dataset, one can predict uptake fluxes from external metabolite concentrations. This neural layer can be reused by any FBA user to improve the predictive power of a metabolic model, with an adequate experimental set-up.

Alternative computation methods to surrogate FBA

The first method (Fig. 1), inspired by previous work on signaling networks[19], learns consensual flux branching ratios found in example flux distributions (*i.e.,* the training set) and represents these ratios in the form of a weight matrix. Since the mass conservation law is the central rule when satisfying metabolic networks constraints, these ratios play a key role in the determination of the metabolic phenotype, *i.e.*, the paths taken by metabolites in the organism. In this approach, we assume that the flux branching ratios remain similar between flux distributions with different bounds on the uptake fluxes. In other words, we postulate that most of the flux distribution examples of our training set resemble a consensual metabolic state, with consensual branching ratios. Consequently, the method

learns from example flux distributions, a weight matrix representing these consensual branching ratios. A simple toy model network is shown to demonstrate the functioning of this first method in Fig. 1a. With the usual FBA method, we obtain the flux distribution maximizing the flux of the v_3 reaction (representing a classical 'biomass' reaction) with an uptake reaction v_1 (representing a classical uptake reaction) by solving a linear program (Fig. 1b). At steady state, metabolite production fluxes can be calculated from reaction fluxes via the transition matrix $P_{v\rightarrow m}$ from fluxes to metabolites, and similarly, reaction fluxes can be calculated from metabolite production fluxes using the matrices $P_{m\rightarrow v}$ and W_r (Fig. 1c). For a given metabolite *j* and a reaction *i*, the weight w_{ji} (in matrix W_r) represents the fraction of metabolite *j* production flux going to reaction *i*.

To learn the weights w_{ji}, one first needs to translate the model into a neural-network-like architecture (Fig. 1d). Precisely, in Fig. 1d we start from an initial set of given fluxes and then propagate knowledge about the fluxes through the entire network, each layer corresponding to one step in a discrete flux propagation. Mathematically, each layer is composed of two simple operations that update the *M* and *V* vectors, respectively representing metabolites production fluxes and reaction fluxes. Those operations are repeated until convergence, with v_1 being constant because it is known from the start (*i.e.*, the input of the network). The network shown in Fig. 1d is the unrolled representation of the RNN (named AMN-Wt) depicted in Fig. 1e. As detailed in the Method section – AMN architectures and parameters, the matrix W_r can be learned through training, assuming the learned weights are a consensus of the different W_r matrices observed in the training set. The steady state fluxes of AMN-Wt iterated 30 times (or more, here we stopped at 30 because the reference data values were reached) equal to those obtained in the reference data (Fig. 1f).

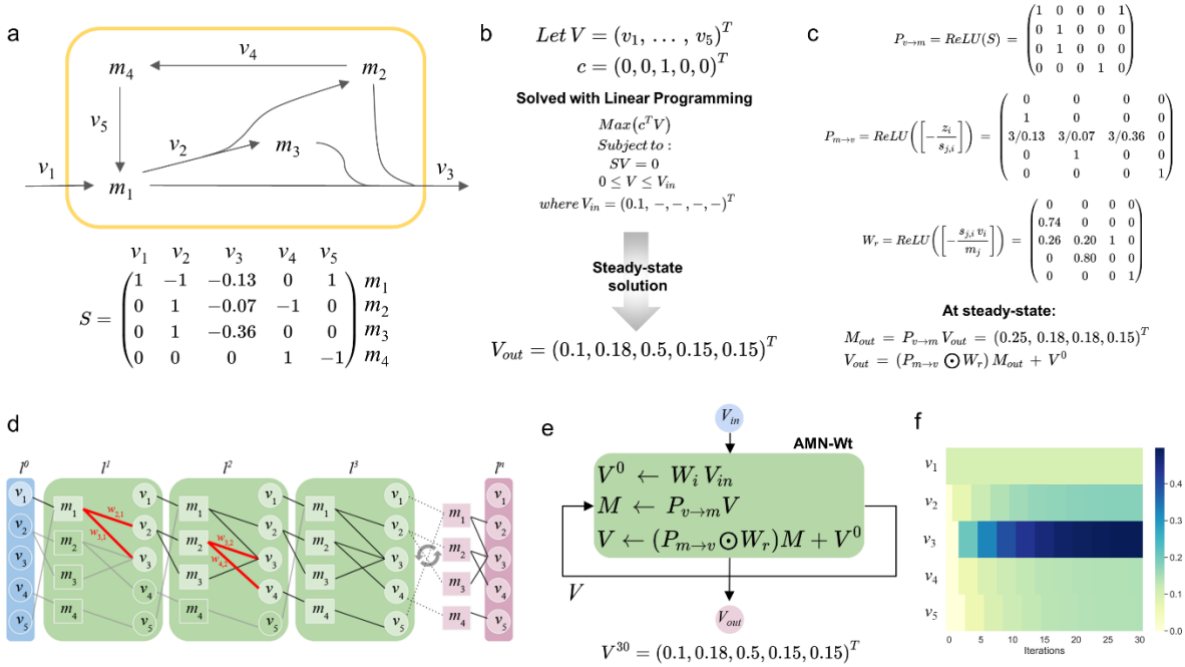

Fig. 1. Computing steady-state fluxes with stoichiometric and neural models. ***a.*** *Simple toy stoichiometric model. The model is unidirectional, all flux values are positive.* ***b.*** *Steady-state solution fluxes maximizing v_3. At steady state, the reaction fluxes (v_i) must satisfy stationarity conditions that guarantee mass balance of all metabolites, this is depicted by the equation $SV = 0$, where S is the stoichiometric matrix representing the connectivity of the model and V the vector of fluxes to be calculated. V_{in} is the medium represented by a vector of nutrient uptake fluxes (here v_1= 0.1, symbol "—" indicates that no value is provided, in practice we use an 'infinity' value to represent an unbounded flux). The steady-state solution V_{out} is calculated by solving a linear program maximizing the objective $c^T V = v_3$, here we used the Cobrapy package[22] to compute V_{out} (simplex-based algorithm).* ***c.*** *Stoichiometric model matrices. $P_{v\to m}$ is the matrix to compute metabolite production fluxes from reaction fluxes, $P_{m\to v}$ is a matrix to compute reaction fluxes from the production fluxes of the reaction substrates. When a metabolite is the substrate of several reactions, each reaction gets a fraction of the metabolite production flux, this is depicted in matrix W_r (r indicates this matrix is used in recurrence). M_{out} is the vector of metabolite production fluxes at steady state, the operator $\odot$ stands for element-wise matrix product (Hadamard product), and $ReLU(x) = max\ (0, x)$. $s_{j,i}$ corresponds to the value of S at the jth row (metabolite) and i^{th} column (flux) and z_i is the number of negative elements in column i of S.* ***d.*** *Unrolled neural network built from the model. In this theoretical example an initial flux vector is set, representing only the uptake fluxes, and is mapped to a flux vector covering the entire network. In the initial layer (l^0) only v_1 has a value (v_1^0). In layer 1, v_1 value is passed to m_1, the production flux for metabolite 1, $m_1^1 = v_1^0$. Subsequently a fraction (w_{21}) of m_1 goes to v_2 and the other fraction (w_{31}) to v_3, therefore $v_2^1 = w_{21} m_1^1$, $v_3^1 = w_{31} m_1^1$ with w_{21}+$w_{31} = 1$. Additionally, v_1 remains as in l^0: $v_1^1 = v_1^0$. In layer 2, v_1 continues to feed m_1, v_2 is passed on to m_2 and m_3 , therefore, $m_2^2 = v_2^1$ and $m_3^2 = v_2^1$, m_2 then goes to v_3 and v_4, and as in the previous layer we have $v_3^2 = w_{32} m_2^2$, $v_4^2 = w_{42} m_2^2$ with w_{32}+ $w_{42} = 1$, other fluxes remain the same as in l^1. In layer 3, m_4 receives input from v_4 which in turn activates v_5. In layer 4, $m_1$receives input from both v_1 and v_5: $m_1^4 = v_1^3 + v_5^3$.* ***e.*** *Recurrent neural network representation. At each iteration step V^l and M^l are computed using matrices $P_{v\to m}$ and $P_{m\to v}$ of panel c, while the matrix W_r can be learned by training with experimental data or model simulations. For example, setting v_1^0 = 0.1 and searching weights for which v_3^n = 0.5 one finds w_{21} = 0.74, w_{31} = 0.26, w_{32} = 0.20, and w_{42} = 0.80. Taking these weights, the V^n values obtained for n=30 match those of panel B.* ***f.*** *Heatmap for flux values up to 30 iterations.*

While the AMN-Wt method is simple to implement it suffers from a drawback. As we shall see later in Table 1, a consensus set of weights leads to a solution when upper bounds (UB) for uptake fluxes are provided, but not when exact bounds (EB) for uptake fluxes are given. Supplementary Fig. S2 shows an example in which two flux distributions, with different EBs on uptake fluxes, lead to different weights for the same networks. Consequently, we cannot assume that AMN-Wt can handle all possible flux distributions in the EB case. To overcome this shortcoming, we next present two alternative methods for solving FBA optimization problems that can accommodate both EB and UB cases for uptake fluxes.

These methods are much closer to the LP or QP optimizations behind FBA, their input can be exact or upper bounds on fluxes. Let us note here that the alternative methods described next are not AMNs *per se*, but are mechanistic models (MM) with gradient backpropagation compatibility that '*replace*' the Simplex-based LP or QP solving algorithms in our AMNs. Importantly, the two methods address two important tasks in flux modeling: optimizing a flux distribution for maximal biomass rate (in LP), as in classical FBA, and fitting a stationary flux distribution to partial flux data (in QP).

The first FBA solver (LP solver, Fig. 2b), derived from a method proposed by Yang *et al.*[23], handles linear problems using exact constraint bounds (EBs) or upper bounds (UBs) for uptake fluxes (V_{in}). That method make use of RNNs for optimization, which is a long-standing field of research[24] inspired by the pioneering work of Hopfield and Tank[25]. Later, these RNNs were showcased to perform well for solving linear programs[26] and simpler and more efficient solutions were developed over the years[23,27]. It is important to point out at this stage that these RNNs are non-trainable networks and differ from the contemporary RNNs used in ML. The RNNs developed for optimization are instead recurrent procedures iteratively updating the solution of linear programs. Nonetheless, as we shall see thereafter, the LP solver of Fig. 2b can backpropagate a gradient and can therefore be connected to trainable layers.

As the AMN-Wt method, the LP solver, iteratively computes fluxes to come closer to the steady-state solution (V_{out}). However, calculations are more sophisticated than for the AMN-Wt, the method integrates the same objective function (c) than the classical FBA Simplex solver and iteratively update the flux vector (V), and the vector (M) representing the dual problem variables also named metabolites shadow prices[28] (*cf.* Method - LP-solver - for further details). To assess the validity of the LP solver, we calculated 100 different growth rates for *E. coli* core model[29] taken from the BIGG database[21] using the Simplex solver available in the Cobrapy package[22]. The *E. coli* core model[29] contains 154 reactions, 72 metabolites (after duplicating bidirectional reactions), including 20 metabolites, which can be imported to the cell through uptake reactions. We generated 100 different uptake fluxes combinations, varying the 20 uptake fluxes of *E. coli* core model (*cf.* Method - Generating Training sets - for further details). Results presented in Fig. 2d exhibits almost identical results as the reference data, with the same initialization, both for known uptake fluxes with EB and for unknown uptake fluxes with UB.

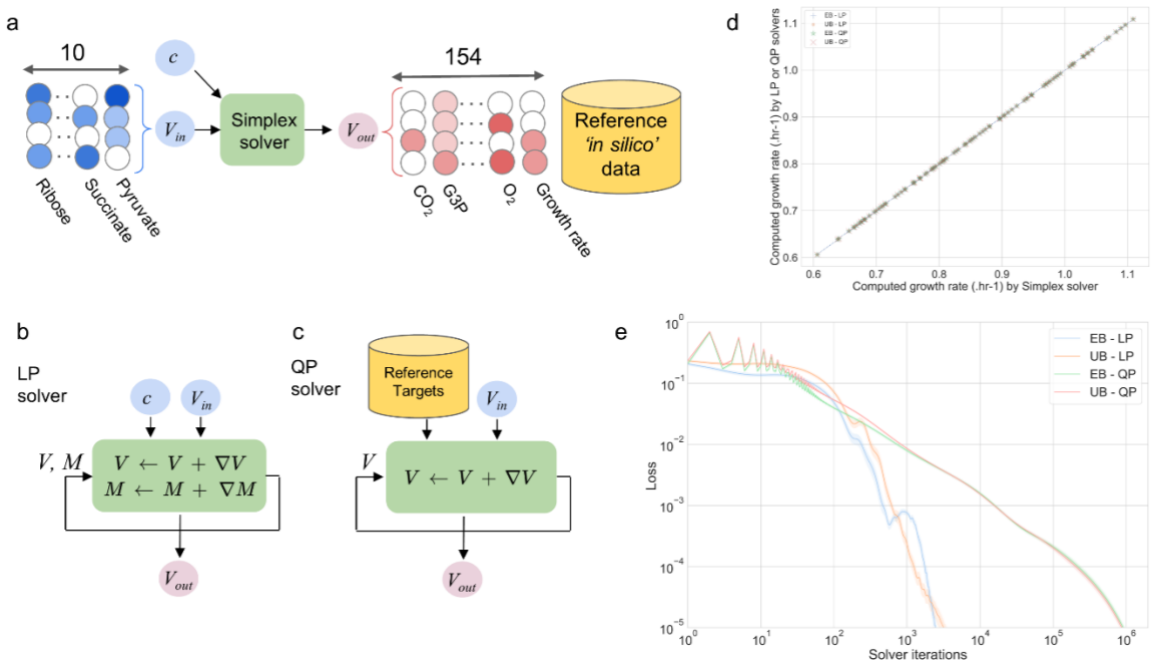

Fig. 2. FBA solver architectures and performances. ***a.*** *Schematic procedure for the Simplex solvers. From V_{in}, which is a vector describing the bounds of some uptake fluxes, the solvers reach a steady-state solution (V_{out}) optimizing the objective function c, satisfying the constraints and bounds of the network (cf. Fig. 1 panel b). One of the most popular solvers is a simplex-based method in Cobrapy[22], taken as reference data for the results shown here.* ***b.*** *Schematic for LP-solver architecture. This in-house solver surrogates the simplex-based algorithm. Following Yang et al.[23] and as further detailed in the Method section (LP-solver Architecture), the full flux distribution V is updated by ∇V and the metabolites shadow prices M by ∇M through products of matrices derived from the stoichiometry of the network.* ***c.*** *Schematic for QP-solver architecture. Here target reference fluxes are given to the solver. The computed fluxes are fitted to the Reference Targets by the means of a custom loss function integrating also the input constraints along with the stoichiometric constraint of the metabolic network. The flux vector V is updated by ∇V which is the gradient minimizing the loss function (cf. Method section - QP Architecture - for further details).* ***d.*** *Matching of growth rate between the Simplex solver and the LP/QP solvers. EB refers to "Exact Bound", a set-up in which V_{in} is composed of the exact flux values for each uptake flux. UB refers to "Upper Bound", in which V_{in} is composed of upper threshold values for each uptake flux.* ***e.*** *Loss vs. Solver iterations. QP takes 1 million iterations to reach close to zero values, whereas LP takes 10,000 iterations. Plotted is the mean and standard error (95% confidence interval) across all elements of the set of 100 simulations.*

The second solver is loosely inspired by the work on Physics-Informed Neural Networks (PINNs), which has been developed to solve partial differential equations matching a small set of observations[30]. With PINNs, solutions are first approximated with a neural network and then refined to fulfill the constraints imposed by the differential equations and the boundary conditions. Refining the solutions necessitates the computation of three loss functions, the first is related to the observed data, the second to the boundary conditions and the third to the differential equations. As detailed in the Method - Loss functions derivation and QP solver - sections, we similarly compute losses for the measured fluxes, the EB or UB boundary constraints, and the metabolic network stoichiometry and flux positivity constraints. As in PINN we next compute the gradient on these losses to refine the solution vector *V*. Unlike with the LP solver, in the present case we do not provide an objective (*c*) to maximize, but the actual targeted measured data (V_{out}), consequently the method is named QP

because it is equivalent to solving an FBA problem with a quadratic program. Our QP solver reaches acceptable losses as the LP one (Fig. 2d) but requires more iterations.

AMNs: metabolic & neural hybrid models for predictive power with mechanistic insights

While the LP and QP solvers perform well, their main weakness is the number of iterations needed to reach satisfactory performances, more than 10,000 (Fig. 2e). Since our goal is to integrate such methods in a learning architecture, this drawback has to be tackled. As illustrated in Fig. 3a, our solution is to improve our initial guesses for fluxes, by training a prior network (a classical dense ANN architecture) to compute initial values for all fluxes (V^0) from exact or upper bounds on uptake fluxes (V_{in}). This prior network is trained along with the LP or QP solvers with the goal of finding V^0 values that are as close as possible to the optimum flux values (V_{out} in the reference data set). This is achieved by setting the iteration number of the LP or QP solvers to low values.

In the remainder of the paper, we name Artificial Metabolic Network (AMN), the hybrid model shown in Fig. 3a composed of a neural network followed by a LP or QP solver. AMN results are represented in Fig. 3b with prior networks of different hidden layer sizes. We note that the solver acts as a mechanistic layer as it contains the stoichiometry and bounds of a metabolic network, whereas the prior network acts as a neural layer that can be trained.

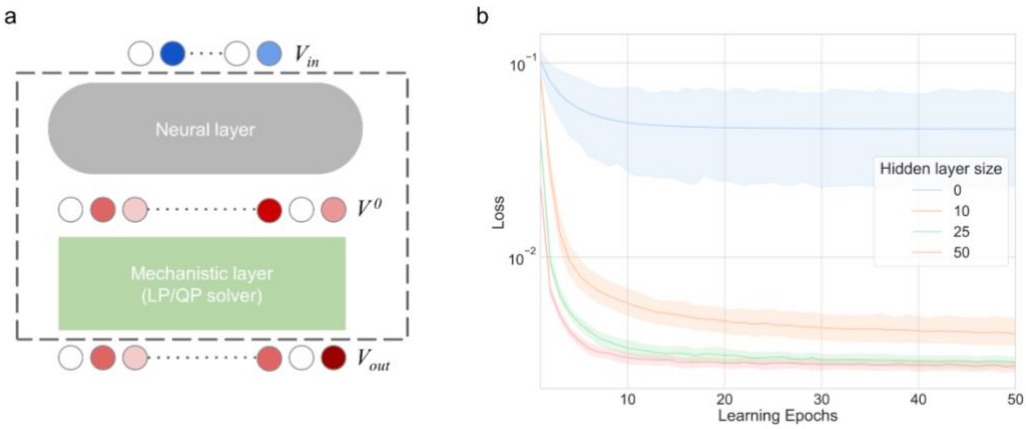

Fig. 3: AMN architecture enabling training. a. *Basic AMN architecture principle. V_{in} is the vector representing upper or exact bounds on uptake fluxes. Each white or blue circle represents one reaction. V^0 contains the initial fluxes predicted by a dense neural network. V_{out} is the final flux vector, computed by the LP or QP solvers.* ***b.*** *Loss vs. Learning Epochs number for E. coli core model[29]. The architecture of panel a (with 4 iterations of the QP solver as the mechanistic layer, with upper bounds) was trained for 50 epochs, measuring the loss on V_{out} after each epoch. Plotted is the loss mean and standard error (95% confidence interval) over a training set of 1000 examples generated with the E. coli core model[29] (see Methods - Generation of training sets with FBA). The different curves correspond to different hidden layer sizes used between V_{in} and V^0.*

The performances of all AMN architectures are given in Table 1 using FBA simulated data on two different *E. coli* metabolic models, *E. coli* core model[29] and iML1515[20] . These models are composed respectively of 154 reactions involving 72 metabolites, and 3682 reactions involving 1877 metabolites

(after duplicating bidirectional reactions). In both cases the Simplex-based solver of Cobrapy[22] was run to optimize growth rates for different media. Each medium was composed of metabolites found in M9 (minimal media) and additional metabolites (carbon sources, amino acids) crossing the cell membrane.

Table 1. AMNs performances. *(1) All SBML models describing different E. coli metabolic models were downloaded from the BiGG database[21], 'Core' stands for the E. coli core model[29], EB (UB) stands for exact bounds (upper bounds) for medium uptake fluxes, the iML1515[20] model was reduced following the procedure described in Methods - 'Making metabolic models suitable for neural computations'. (2) Training set size and range for the number of metabolites added to M9 growth medium, and method used when running Cobrapy[22]. (3) YES or NO if the model contains a neural layer or a mechanistic layer. (4) MM stands for Mechanistic Model, AMN stands for Artificial Metabolic Network. Architectures are described in Methods - 'AMN architectures and parameters'. (5) Number of trainable parameters and epochs, in all cases dropout = 0.25, batch size = 5, the optimizer is Adam and the loss function is the mean squared error between predicted and provided fluxes to which loss constraints are added, see Methods - 'Loss functions derivation' for additional details. (6) Regression coefficient and Loss values for training set (R^2), and cross-validation sets (Q^2) between provided growth rate and predicted growth rate. (7) Regression coefficient and Loss values for growth rates for independent test sets not seen during training. Test set sizes are 10% of training set sizes. For (6) and (7) the performance is displayed as the mean over 5 folds (or over a training set when no cross-validation scheme is performed, i.e., for the MM performances).*

SMBL model Bound (1)	Size Range (2)	Neural layer Mechanistic layer (3)	Architecture Timestep (4)	Nbr param. Nbr epochs (5)	Training R^2 Loss constraint (6)	5-fold Q^2 Loss constraint (6)	Test set Q^2 Loss constraint (7)
Core	100	NO	MM_LP	n/a	1.000 ± 0.000	n/a	n/a
EB	1-6	YES	10^4	n/a	3.2e-9 ± 3.2e-8	n/a	n/a
Core	100	NO	MM_LP	n/a	1.000 ± 0.000	n/a	n/a
UB	1-6	YES	10^4	n/a	5e-7 ± 2.8e-6	n/a	n/a
Core	100	NO	MM_QP	n/a	1.000 ± 0.000	n/a	n/a
EB	1-6	YES	10^6	n/a	7.8e-6 ± 6.1e-6	n/a	n/a
Core	100	NO	MM_QP	n/a	1.000 ± 0.000	n/a	n/a
UB	1-6	YES	10^6	n/a	7.1e-6 ± 5.7e-6	n/a	n/a
Core	1000	YES	AMN_LP	17 808	0.98 ± 7.9e-3	0.98 ± 7.4e-4	0.98
EB	1-6	YES	4	500	2.8e-3 ± 0.6e-3	2.8e-3 ± 0.5e-3	3.0e-3
Core	1000	YES	AMN_LP	25 152	0.98 ± 9.7e-3	0.97 ± 1.0e-2	0.99
UB	1-6	YES	4	500	2.5e-3 ± 0.4e-3	2.5e-3 ± 0.4e-3	3.1e-3
Core	1000	YES	AMN_QP	8904	0.99 ± 4.2e-3	0.99 ± 4.7e-3	0.98
EB	1-6	YES	4	500	2.3e-3 ± 0.5e-3	2.3e-3 ± 0.5e-3	3.0e-3
Core	1000	YES	AMN_QP	8904	0.97 ± 9.9e-3	0.97 ± 1.3e-2	0.97
UB	1-6	YES	4	500	2.5e-3 ± 0.6e-3	2.5e-3 ± 0.6e-3	2.0e-3
Core	1000	YES	AMN_Wt	13 622	0.99 ± 1.3e-3	0.99 ± 2.2e-3	1.0
UB	1-6	YES	4	500	0.9e-3 ± 0.000	0.9e-3 ± 0.000	0.000
iML1515	11000	YES	AMN_LP	839 266	1.0 ± 1.0e-3	1.0 ± 1.0e-3	1.0
UB	1-4	YES	4	100	0.000 ± 0.000	0.000 ± 0.000	0.000

iML1515	11000	YES	AMN_QP	295 050	1.0 ± 1.4e-3	1.0 ± 1.4e-3	1.0
UB	1-4	YES	4	100	0.000 ± 0.000	0.000 ± 0.000	0.000
iML1515	11000	YES	AMN_Wt	634 238	1.0 ± 0.1e-3	0.99 ± 0.4e-3	1.000
UB	1-4	YES	4	100	0.000 ± 0.000	0.000 ± 0.000	0.000

All architectures presented in Table 1 exhibit excellent regression coefficients and losses for training sets, validation sets and test sets, and this for both models *E. coli* core[29] and iML1515[20]. The MM architectures (without neural layers) can compute losses for metabolic constraints (boundary and stoichiometry) but cannot be used to make predictions on cross-validation or test sets. The AMN architectures can be used to both make predictions (Q^2) and compute losses. It is interesting to observe the good performances of AMN-Wt when UB training sets are provided. Indeed, while counterexamples can be found for which AMN-Wt will not work with EB training sets (*cf.* Fig. S2), we argue in Supplementary Information – AMN-Wt architecture – that AMN-Wt is able to handle UB training sets because the initial inputs (UB values for uptake fluxes) are transformed into suitable exact bound values during training. Yet, the consensual weight matrix (W_r in Fig. 1 or Fig. S1) calculated during training does not have a direct physical meaning. Weights in W_r arise at branch point metabolites that are consumed by at least two reactions (like m_1 and m_2 in Fig. 1). Consequently, the weights should correspond to flux split ratios, for instance, taking the example of Fig. 1, the metabolite production flux m_1 is spliced into $w_{21}m_1$ and $w_{31}m_1$. We show in Fig. S3, that the flux split ratios are conserved for nutrients leading to different metabolite production fluxes if the Michaelis-Menten kinetics parameters (V_{max} and K_m) of the enzymes catalyzing the reactions involved in the split remain constant. However, Chubukov *et al.*[31] have shown experimentally that it was not the case (for *B. subtilis*) and different nutrients do provide different ratios. This behavior is due to varying enzyme activities, which themselves depend on enzyme concentrations, post-translational modification, and gene regulations. Chubukov *et al.*[31] clearly show with experimental evidence that different nutrients give rise to different concentrations for many enzymes, implying that nutrients do have an effect on gene regulations.

In classical FBA, all regulations explaining a given flux distribution are completely ignored, and the flux distribution computation entirely relies on manually setting bounds on uptake fluxes. Therefore, when performing classical FBA, one needs to consider each condition independently from the other, to reach different metabolic phenotypes. Similarly, AMNs attempt to take regulations into account in the neural layer, while keeping the mechanistic layer for metabolic phenotype computations. However, unlike in FBA, AMNs attempt to learn the relationship between a set of flux bounds (on uptake reactions) and

the whole steady-state metabolic phenotype. As a result, AMNs are generalizing this relationship for a set of conditions, and not just one as in FBA.

We note that even though weights do not have a physical meaning, AMN-Wt stills exhibits excellent performances, showing that the consensual W_r matrix and the initial V^0 vector (computed through a neural layer from the upper bound V_{in}) are performant enough. We also note that the weight issue does not arise with AMN-LP and AMN-QP as these architectures do not rely on flux split ratios.

AMNs can be directly trained on experimental datasets with good predictive power

To train AMNs on an experimental dataset, we grew *E. coli* DH5-alpha in 110 different media compositions, with M9 supplemented with 4 amino acids as a basis and 10 different carbon sources as possibly added nutrients. From 1 up to 4 carbon sources were simultaneously added in the medium at a concentration of 0.4 $g.L^{-1}$ (more details in Methods - Culture conditions). We determined which compositions to test by choosing all the 1-carbon source media compositions and randomly picking one hundred of the 2-, 3- and 4-carbon sources media compositions (more details in Methods - Generation of an experimental training set). The growth of *E. coli* was monitored in 96-well plates, by measuring the optical density at 600nm (OD_{600}) over 24 hours. The raw OD_{600} was then passed to a maximal growth rate determination method, based on a linear regression performed on $\log(OD_{600})$ data (more details in Methods - Growth rate determination).

The experimental dataset was used to train all AMN architectures (-LP, -QP, -Wt). For AMN -LP/-QP we used the architecture plotted in Fig. 3, for AMN-Wt we used the architecture of Fig. S1. In all cases the mechanistic layer was derived from the stoichiometric matrix of the iML1515[20] *E. coli* reduced model (*cf*. Method section - Making metabolic networks suitable for neural computations). For all AMNs upper bounds for nutrient uptake fluxes' upper bounds were provided, as exact values remain unknown. Results are provided in Fig. 4.

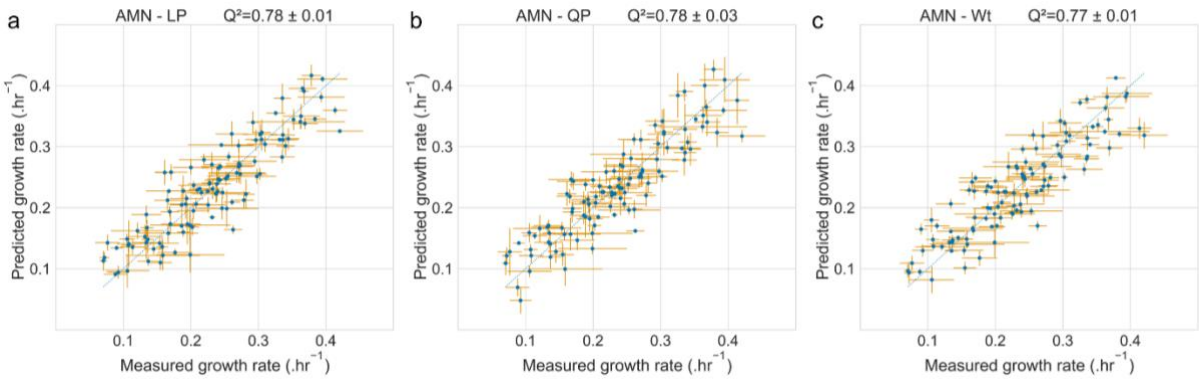

Fig. 4: Benchmarking growth rate predictions by AMNs with experimental measurements*. In all panels the experimental measurements were carried on E. coli grown in M9 mixed carbon sources (strain DH5-alpha, model iML1515[20]). Training and 10-fold stratified cross-validation were performed 3 times with different initial random seeds each for 1000 epochs. All points plotted correspond to predicted values not present in the training set. This was compiled using all predicted values obtained for each cross-validation set. In all cases, means are plotted for both axes (measured and predicted), error bars are standard deviations. Supplementary File 'Data_Fig4.xlsx' compiles raw results used for this figure (see Data availability).* ***a.*** *Performance of the AMN-LP architecture. The neural layer of AMN-LP is composed of an input layer of size 38, an hidden layer of size 500, and an output layer of size 550 corresponding to all fluxes and 1083 metabolite shadow prices of a reduced iML1515 model[20]. The mechanistic layer takes as input the 550+1083 outputs of the neural layer and minimizes the loss between measured and predicted 'biomass' reaction fluxes and the losses of the metabolic network constraints.* ***b.*** *Performance of the AMN-QP architecture. The neural layer of AMN-QP is composed of an input layer of size 38, an hidden layer of size 500, and an output layer of size 550 corresponding to all fluxes of the iML1515[20] reduced model.* ***c.*** *Performance of the AMN-Wt architecture. The neural layer of AMN-Wt (matrix W_i in Figure S1) is composed of an input layer of size 38, an hidden layer of size 500, and an output layer of size 550 corresponding to all fluxes of the iML1515[20] reduced model, the size of the recurrent W_r matrix of Fig. S1 is 550x550.*

AMNs can be used in a reservoir computing framework to enhance the predictive power of traditional FBA solvers

Once an AMN has been trained on a large dataset of simulated data, we can fix its parameters and exploit it in subsequent further learning. As already mentioned in the introduction section, the uptake fluxes of *E. coli* nutrients, as well as their relation to external concentrations, remain largely unknown: the quantitative rate for each compound may vary between growth media, unlike in classical FBA calculations, where the same upper bound (or zero, if a compound is absent) is used in all cases. This is a common flaw of FBA, that is only partially remedied in satFBA[18] or mcFBA[5], as those relies on supplementary knowledge (and additional experimental measurements). To use the architectures of Fig. 3, we thus need to first convert nutrient concentrations into uptake fluxes. This is achieved by adding a neural layer that precedes the AMN (Fig. 5a). In this architecture the AMN is no longer trainable, only the prior neural layer is, and the AMN acts as a reservoir as in reservoir computing[32]. Among the various architectures we benchmarked, we took as a reservoir the AMN-QP of Table 1 trained on iML1515[20] UB dataset.

The predictive power of our AMN-reservoir was assessed by a regression coefficient $R^2 = 0.97$ (Fig. 5c) on training set, and $Q^2 = 0.76$ (Fig. 5b) on a test set built from aggregated validation sets during 10-fold

cross-validation. To compare with an 'out-of-the-box' FBA approach, we first used an arbitrary value for the present compounds, and a zero value for the absent compounds. Applying these values on the corresponding uptake fluxes upper bounds, and optimizing the 'biomass' reaction flux produced results poorly correlated with experimental measurements. To enhance these results, we searched optimized upper bound values for uptake fluxes (see Methods - Optimization of uptake fluxes upper bounds in FBA). It resulted in a R^2 = 0.51 performance (Fig. 5f), showing a much weaker fit than our AMN-reservoir. Overall, our results indicate that AMNs can be used to substantially increase the predictive capabilities of FBA without costly experimental work.

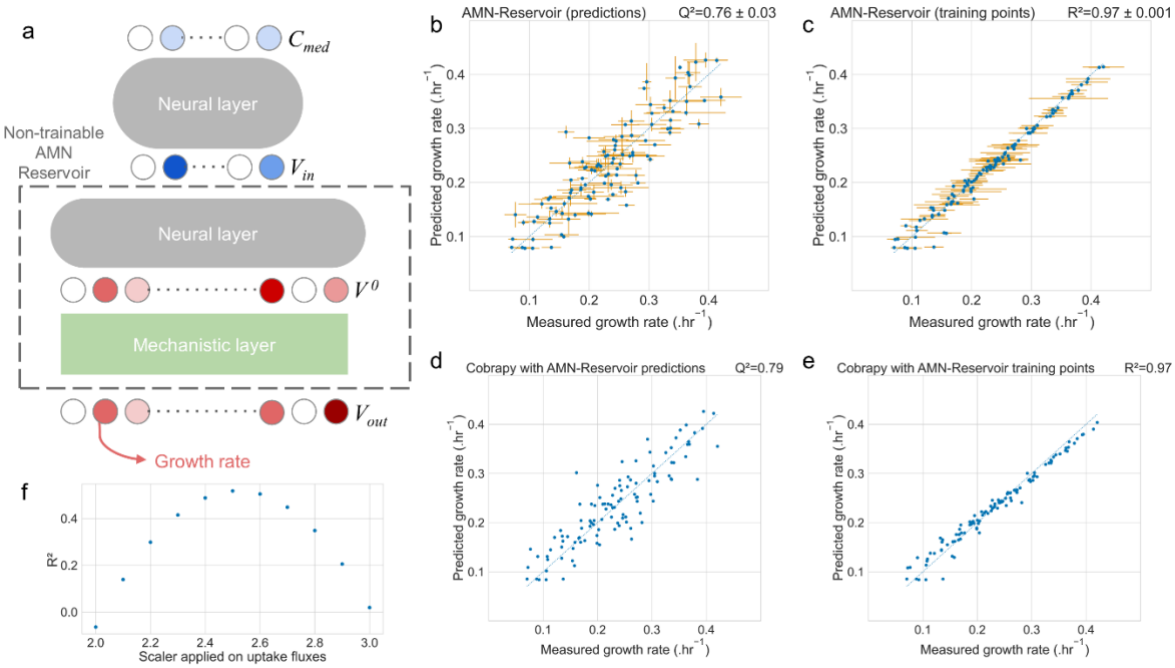

Fig. 5: Reservoir computing for improving the predictive power of FBA modeling *(strain DH5-alpha, model iML1515[20]).* ***a.*** *AMN used in reservoir computing. The non-trainable AMN-reservoir (dotted gray box) is the AMN shown in Figure 3.b (AMN-QP trained on iML1515 with UB bounds with performances given in the second to last row of Table 1) and is connected to a prior trainable network which purpose is to compute medium uptake fluxes (V_{in}) from the concentration (C_{med}) of metabolites added to medium. Taking as input V_{in}, the non-trainable reservoir prints out all fluxes including the growth rate.* ***b.*** *Performance of the reservoir computing model shown in panel a, in terms of predicted growth rate vs. measured growth rate. As in Fig. 4, all points plotted correspond to predicted values not present in the training set. This was compiled using all predicted values obtained for each validation set in 10-fold stratified cross validation repeated 3 times with different initial random seeds.* ***c.*** *Performance of the AMN-reservoir with training points instead of predictions. All points plotted correspond to computed values in the same cross-validation scheme as in panel b, using the points of the training sets instead of validation sets.* ***d.*** *Performance of Cobrapy[22] when extracting V_{in} from panel b predictions to be used as inputs.* ***e.*** *Performance of Cobrapy[22] when extracting V_{in} from panel c computations to be used as inputs.* ***f.*** *R^2 between measured growth and computed growth by Cobrapy[22], using different scalers on C_{med} to compute V_{in}. Here, Cobrapy[22] was run taking as input upper bound uptake fluxes for added metabolites in medium. Intake flux values were set to , and then scaled (0 when the metabolite was absent in the medium and 1*scaler when it was present - see section Method, Optimization of uptake fluxes upper bounds in FBA, for details on optimization procedure)*

Discussion and conclusion

In this study, we showed how a neural network approach, with metabolic networks as a learning architecture, can be used to address metabolic modeling problems. Previous work on RNNs and PINNs for solving constrained optimization problems was re-used and adapted to develop three models (AMN-Wt, -LP and -QP) enabling gradient backpropagation within metabolic networks. The models exhibited excellent performances on FBA generated training sets (Table 1). We also demonstrated the models could directly be trained on an experimental *E. coli* growth rate dataset with good predictive abilities (Fig. 4). For improved scalability and adaptability, we trained an AMN-reservoir on a large FBA generated training set, and used the reservoir to improve FBA predictions on the experimental dataset. Figure 5 shows that our AMNs far outperform the results obtained by the simplex-based Cobrapy[22] FBA solver.

Determining uptake fluxes is a core experimental work required for making FBA predictions realistic. Here, we developed approaches that get rid of such needs for reaching plausible fluxes distributions. We did so by backpropagating the error on the growth rate, to find complex relationships between the medium concentrations and the medium uptake fluxes. To this end, we demonstrated the high predictive power of AMNs, and their re-usability in classical FBA approaches. Indeed, FBA developers and users may now make use of our AMN-reservoir method for relating medium uptake fluxes to growth medium concentrations. In this regard, Supplementary File 'Data_Fig5.xlsx' (see Data availability) gives uptake fluxes for the metabolites used in our benchmarking work with *E. coli* (Fig. 5), these upper bounds for uptake fluxes that can directly be used by Cobrapy[22] to reproduce Fig. 5d and 5e.

Making FBA suitable for training like we have done in this paper opens the door to improve FBA models. For instance, in addition to uptake fluxes, AMNs could also be used to search for the coefficients of the biomass reaction appropriate to best fit measurements. So far, these coefficients are derived based on literature, but also using experimental data: growth rate, flux, and macromolecular fractions measures can help finding optimal coefficients[20]. However, these experiments are limited in number and performed once for the reaction parametrization, in a single experimental setup, meaning these coefficients are hardly extrapolated to all possible conditions. Some studies already underline this issue and attempt to efficiently integrate experimental data in the biomass reaction parametrization[33]. With AMNs, a trainable layer containing the reaction's coefficients could be added, adapting the biomass reaction to any experimental setup.

Another task that AMNs could handle is gene regulation which is not explicitly taken into account in classical FBA. Here a set of genes (including enzymes) corresponding to operons could be encoded via trainable layers either turned on or off. Such AMNs could be trained on a variety of experimental inputs (wider than the carbon source composition as shown in this study) to grasp the complexity of the regulation processes happening in the cell to better explain the end-point metabolic steady-state phenotype of the organism.

Returning back to the curse of dimensionality issue mentioned in the introduction, we systematically searched training set sizes for which 'black-box' ML methods would yield performances similar to our AMN hybrid models. To that end, we trained a simple dense ANN model on training sets of increasing sizes. Results obtained with *E. coli* core[29] show that at least 500,000 entries are needed in the training sets to reach losses below 10^{-1} (*cf*. in Supplementary information – AMN and ANN training set sizes and Fig. S7), which according to Table 1 are still 2 orders of magnitude higher than all AMNs losses trained on only 1000 entries. This clearly demonstrates the capacity of hybrid models to reduce training set sizes by constraining the search space through the mechanistic layer.

Beyond FBA and black-box ML improvements, AMNs can also be exploited for industrial applications. Indeed, since arbitrary objective functions can be designed and AMNs can be directly be trained on experimental measurements, AMNs can also be used to optimize media for the bioproduction of compounds of interest or to find the best gene deletion and insertion strategy in typical metabolic engineering projects. In this latter case, a trainable 0/1 layer turning on or off reactions can be added prior to the mechanistic layers of our AMNs. Another potential application is the engineering of microorganism-based decision-making devices for the multiplexed detection of metabolic biomarkers or environmental pollutants. Here, AMNs could be used to search for internal metabolite production fluxes enabling one to differentiate positive samples containing biomarkers or pollutants from negative ones. Such a device has already been engineered in cell-free systems[34], and AMNs could be used to build a similar device *in vivo* by adding a trainable layer after the mechanistic layer which purpose would be to select metabolite production fluxes that best split positive from negative samples.

Methods

Making metabolic networks suitable for neural computations

The set-up of our AMNs requires all reactions to be unidirectional, that is, the solutions must show positive-only fluxes (which is not guaranteed by usual genome-scale metabolic models). To split

reactions of a given metabolic network into separate forward and reverse reactions, we wrote a standardization script that loads an SBML model into Cobrapy[22] and screens for all two-sided reactions, then duplicating them into two separate reactions; and writes a new version of the model with bi-directional reactions split into separate forward and backward reactions. To avoid confusion, we add a suffix for these reactions, either "for" or "rev" respectively designating the original forward reaction and the reversed reaction. The uptake reactions were also duplicated, even if encoded as one-sided, and their suffix was set to "i" for inflow reactions (addition of matter to the system), and "o" for outflow reactions (removal of matter from the system).

As detailed in the next subsection, our unidirectional models are used to build training sets. The duplicated iML1515[20] model is quite large, comprising 3682 reactions and 1877 metabolites. A substantial number of reactions in that model have a zero flux for many different media and it is unnecessary to keep these reactions during the training process. Prior training, we therefore generate a reduced model by removing zero flux reactions along with the metabolites no longer contributing to any reactions. Using that procedure, we were able to reduce iML1515[20] model to only 550 reactions and 1083 metabolites.

Generation of training sets with FBA

Our reference flux data are obtained from FBA simulations, using the GNU Linear Programming Kit ('GLPK', a simplex-based method) on Cobrapy[22], with different models of different sizes. Throughout this paper, when 'reference data' is mentioned, it refers to data computed with this method.

Reference data for metabolic flux distributions were generated using models downloaded from the BiGG database[21]. The models were used to generate data using Cobrapy[22] following a precise set of rules. First, we identified essential uptake reactions for the models we used (*E. coli* core[29] and iML1515[20]). Precisely, if one of these reactions has its flux upper bound set to zero, the 'biomass' reaction optimization is impossible, even if all other uptake fluxes bounds are set to a high value, e.g., 1000. In other words, we identified the minimal uptake fluxes enabling growth according to the models. During reference data generation, the upper bounds on these reactions were set to 10. For *E. coli* core[29] we found 7 of such obligate reactions (for the uptake of CO2, H+, H20, NH4, O2, Phosphate, and Glycerol as the carbon source). For iML1515[20] we had the same 7 obligate reactions and additional salt and ions uptake reactions (for the uptake of Fe2+, Fe3+, Mn2+, Zinc, Mg, Calcium, Ni2+, Cu2+, Selenate, Co2+, Molybdate, Sulfate, K+, Sodium, Chloride, Tungstate, Selenite). With iML1515[20], we also added as obligate reactions the uptake of four amino acids (Alanine, Proline, Threonine and

Glycine) in order to be consistent with our experimental training set where the four amino acids were systematically added to M9 (*cf.* next subsection).

To generate different media compositions, we added to the obligate reactions, a set of variable uptake reactions. For the *E. coli* core model[29] we added 13 variable uptake reactions (for the uptake of Acetate, Acetaldehyde, Oxoglutarate, Ethanol, Formate, Fructose, Fumarate, Glutamine, Glutamate, Lactate, Malate, Pyruvate, and Succinate). For each generated medium, variable uptake reactions were selected following a binomial distribution B(n, p) with n=13 and p = 0.5, p being a tunable parameter related to the ratio of selected reaction. Consequently, the mean number of selected variable uptake reactions was n x p = 6.5. Next for each selected reaction, the upper bound value of the reaction flux was drawn randomly between 2 and 10. For the iML1515[20] model, to limit the combinatorial search space, the selected variable uptake reactions were those of the experimental training set and consequently between 1 and 4 variable uptake reaction were added (*cf.* next subsection). The upper bound values for each selected variable reaction were chosen randomly between 0 and 2.2 (0 excluded). The 2.2 threshold was chosen to produce predicted growth rates that were in the range of those observed experimentally.

After generating the set of media for *E. coli* core[29] and iML1515[20] we ran Cobrapy[22] for each medium with FBA and recorded the steady state value for all fluxes including the growth rate (flux of the 'biomass' reaction). These rates were used as a training set for all models presented in Table 1. For all UB training sets, the variable uptake flux values were those used by Cobrapy[22] to generate the training set. For EB training sets, the variable uptake flux values were those calculated by Cobrapy[22] at steady state.

Generation of an experimental training set

Ten carbon sources (Ribose, Maltose, Melibiose, Trehalose, Fructose, Galactose, Acetate, Lactate, Succinate, Pyruvate) were picked for being the variables of our training sets. These could ensure observable growth as a sole carbon source with a concentration of 0.4 $g.L^{-1}$ in our M9 preparations. The selected carbon sources enter different parts of the metabolic network: 6 sugars enter the upper glycolysis pathway, and 4 acids enter the lower glycolysis pathway or the TCA cycle. With a binary (*i.e.*, presence or absence of each carbon source) approach when generating the combinations to test for making the experimental training set, we generated all possible combinations of 1, 2, 3 or 4 carbon sources simultaneously present in the medium. Naturally, we picked all 1-carbon source media combinations for experimental determination (only 10 points). Then, we randomly selected 100 more combinations to experimentally determine, by randomly picking 20 points from the 2-, 40 points from

the 3- and 40 points from the 4-carbon source combinations sets. The python scripts to generate these combinations and pick the ones for making our experimental training set are available in our Github package (see Codes availability section). After picking the combinations to test, we experimentally determined the maximum specific growth rate of *E. coli* for each combination of carbon sources in M9 (see next two subsections). The mean over replicates for each media composition was computed as the corresponding growth rate value to make the final experimental training set (*cf*. Method - Growth rate determination).

Culture conditions

The base medium for culturing E. coli *DH5-α* (DH5a) was a M9 medium prepared with those final concentrations: 100µM CaCl2, 2mM MgSO4, 1X M9 salts (3 $g.L^{-1}$ KH2PO4, 8.5 $g.L^{-1}$ Na2HPO4 2H2O, 0.5 $g.L^{-1}$ NaCl, 1$g.L^{-1}$ NH4Cl), 1X trace elements (15 $mg.L^{-1}$ Na2EDTA 2H20, 4.5 $mg.L^{-1}$ ZnSO4 7H2O, 0.3 $mg.L^{-1}$ CoCl2 6H2O, 1 $mg.L^{-1}$ MnCl2 4H2O, 1 $mg.L^{-1}$ H3BO3, 0.4$mg.L^{-1}$ Na2MoO4 2H20, 3 $mg.L^{-1}$ FeSO4 7H2O, 0.3 $mg.L^{-1}$ CuSO4 5H2O; solution adjusted at pH=4 and stored at 4°C), 1 $mg.L^{-1}$ Thiamine-HCl and 0.04$g.L^{-1}$ amino acid mix so that each amino acid (L-Alanine, L-Proline, L-Threonine, Glycine) was at a final concentration of 5 $mg.L^{-1}$ in the medium. The additional carbon sources that could be added were individually set to a final concentration of 0.4 $g.L^{-1}$. The pH was adjusted at 7.4 prior to a 0.22µm filter sterilization of the medium. Pre-cultures were recovered from glycerol -80°C stocks, grew in Luria-Bertani (LB) broth overday, then used as inoculate in 200µL M9 (supplemented with variable compounds) in 96 U-bottom wells plates overnight, then passed to a replicate of the plate on the next day. The temperature was set to 37°C in a plate reader (BioTek HTX Synergy), with continuous orbital shaking, allowing aerobic growth for 24 hours. A monitoring every 10 minutes of the optical density at 600 nm (OD_{600}) was performed. A figure for summarizing the experimental workflow is available in Supplementary Fig. S6.

Growth rate determination

The maximal growth rate was determined by sliding a window of 1 hour-size, performing a linear regression on the log(OD_{600}) data in each window. We then retrieve the maximum specific growth rate as the maximum regression coefficient over all windows. If several growth phases are visible, one can omit a part of the growth curve for the maximal growth rate determination (for this study we always retrieved the maximal growth rate on the first growth phase, so as we are certain that the media contains all added carbon sources). 8 replicates for each medium composition were performed (on a single column of a 96-well plate). Outliers were manually removed after visual inspection of the growth curves or clear statistical deviation of the computed growth rate from the remaining replicates. The

number of preserved replicates range from 2 to 8, with an average of 4.6 (± 1.6) replicates per medium composition. The mean and standard deviations over replicates were computed to be used for training AMNs and making figures. The code for processing raw data and all the raw data itself is available in the Github package (see Code availability and Data availability sections).

Loss functions derivation

Loss functions are necessary to assess the performances of all MM (MM-LP, -QP) and AMN architectures (AMN-Wt, -QP, and -LP) and also to compute the gradients of the QP solvers.
To perform the derivation, we assume we have a metabolic model with n reactions and m metabolites. Let $V = (v_1, \dots, v_n)^T$ be the reaction flux vector and S the $m \times n$ stoichiometric matrix of the model. We assume some metabolites can be imported in the model, with each a corresponding uptake reaction. Let V_{in} be the vector of n_{in} upper bounds (or exact values) for these uptake reactions, and let P_{in} the $n_{in} \times n$ projection matrix such that $V_{in} = P_{in}V$. We also assume some reaction fluxes have been experimentally measured, let V_{out} be the vector of measured values. With P_{out} the $n_{out} \times n$ projection matrix for measured fluxes, V is calculated solving the following quadratic program (QP):

$$min(\ \|P_{out}\, V - V_{out}\|^2) \tag{1}$$

subjected to:

$$S\,V = 0$$
$$P_{in}V \leq V_{in}$$
$$V \geq 0$$

For each solution (V) of eq. (1), four losses apply. The first is related to the fit to the reference targeted values and the three additional losses are related to the boundary, stoichiometric and flux positivity constraints of the metabolic network.

The first loss is simply the Mean Square Error (MSE) between predictions (V) and measurements(V_{out}):

$$L_1 = \frac{1}{n_{out}} \|P_{out}V - V_{out}\|^2 \tag{2}$$

The second loss is linked to the network stoichiometric constraint ($S\,V = 0$), which in its normalized form (loss per constraint) is:

$$L_2 = \frac{1}{m}\ \|SV\|^2 \tag{3}$$

The third loss evaluates how well boundary constraints are respected ($P_{in}V \leq V_{in}$):

$$L_3 = \frac{1}{n_{in}} \|ReLU(\, P_{in}V - V_{in}\,)\|^2 \tag{4}$$

The last loss enforces all fluxes to be positive:

$$L_4 = \frac{1}{n} \|ReLU(-V)\|^2 \tag{5}$$

We note that when exact bound are provided, $P_{in}V = V_{in}$, and L_3 becomes obsolete as the values of V corresponding to V_{in} are not updated by the LP/QP solvers and AMN programs.

The total loss (L) is calculated as the mean sum of the previous losses:

$$L = (L_1 + L_2 + L_3 + L_4) \;\; / \, 4N \tag{6}$$

where N is the sample batch size.

L can also be computed as the MSE between the vectors

$$\left(P_{out}V, \frac{\|SV\|}{\sqrt{m}}, \frac{\|ReLU(\, P_{in}V - V_{in}\,)\|}{\sqrt{n_{in}}}, \frac{\|ReLU(-V)\|}{\sqrt{n}}\right) \text{ and } (V_{out}, 0, 0, 0) \tag{7}$$

QP solver

The QP solver solves the quadratic program given by eq. (1) in the Loss functions derivation subsection. While the QP system can be solved by a simplex algorithm, solutions can also be approximated calculating V minimizing the loss of eq. (6):

$$min\left(\frac{1}{n_{out}} \|P_{out}V - V_{out}\,\|^2 \;\; + \frac{1}{m}\|SV\|^2 + \frac{1}{n_{in}}\|ReLU(\, P_{in}V - V_{in}\,)\|^2 + \frac{1}{n}\, \|ReLU(-V)\|^2\right) \tag{8}$$

vector V can thus be found solving:

$$\frac{\partial\left(\frac{1}{n_{out}}\|P_{out}V - V_{out}\,\|^2 \;\; + \frac{1}{m}\|SV\|^2 + \frac{1}{n_{in}}\|ReLU(\, P_{in}V - V_{in}\,)\|^2 + \frac{1}{n}\, \|ReLU(-V)\|^2\right)}{\partial V} = 0 \tag{9}$$

V satisfying (3) can be found iteratively:

$$\begin{aligned} V^{(t+1)} &= V^{(t)} + dt\, \nabla V \\ V^{(0)} &= P_{in}^T V_{in} \end{aligned} \tag{10}$$

where t is the iteration number, dt the learning rate, and:

$$\nabla V = \frac{2}{n_{out}}\, P_{out}^T (P_{out}\, V - V_{out}) + \frac{2}{m}\, S^T SV + \frac{2}{n_{in}} P_{in}^T\, D_{in} ReLU(P_{in}V - V_{in}) - \frac{2}{n}\, D_V\, ReLU(-V) \tag{11}$$

where $D_{in} = \frac{ReLU(P_{in}V - V_{in})}{ReLU(P_{in}V - V_{in})}$ and $D_V = \frac{ReLU(-V)}{ReLU(-V)}$ using an *Hadamard* division: $\frac{A}{A} = \left(\frac{a_{ij}}{a_{ij}}\right)$ ($= 0$ when $a_{ij} = 0$)

When exact bounds are provided (EB case), the gradient becomes:

$$\nabla V = \left(\frac{2}{n_{out}} P_{out}^T (P_{out} V - V_{out}) + \frac{2}{m} S^T S V - \frac{2}{n} D_V \, ReLU(-V) \right) \odot \left(1_{n\times 1} - P_{in}^T 1_{n_{in}\times 1} \right) \quad (12)$$

where $\odot$ stands for *Hadamard* product and $1_{n\times 1}$ $\left(1_{n_{in}\times 1}\right)$ is a constant (=1) vector of dimension n (n_{in})

Details on gradient calculations can be found in the Supplementary Information – section QP solver equations.

LP solver

When uptake fluxes are known (EB method), the linear optimization problem solved with FBA for a metabolic network with *n* fluxes and *m* metabolites can be written as:

$$Max: c^T V \quad (13)$$

$$st:$$
$$SV = b$$
$$V \geq 0$$

and its dual form:

$$Min: b^T M$$
$$st: S^T M \leq c^T \quad (14)$$

In the case where only uptake fluxes upper bounds are known (UB method) it can be written as:

$$Max: c^T V$$

$$st: S_{iq} V \leq b \; and \; S_{eq} V = 0 \quad (15)$$
$$V \geq 0$$

its dual form being eq. (14).

In eqs. (13-15), *V* is the vector of fluxes, *M* its dual, referred to as metabolite shadow prices, *S* the stoichiometric matrix, *b* a vector of dimension *m* with b_i corresponding to input fluxes of medium metabolites and *c* the objective vector of dimension *n*. In the UB case, S_{iq} is the stoichiometric matrix involving only internal reactions and S_{eq} is the stoichiometric matrix with uptake reactions. Both matrices that can be derived from *S* are given in Figs. S3 and S4 in Supplementary Information. Keeping in mind that *b* takes positive values only for metabolites transported by uptake reactions, consequently $b_i = 0$ for all internal metabolites. Using the notation of Fig. 1 we note that $b = P_{m\rightarrow v} V^0$, where V^0 is the vector of uptake fluxes. In this work *c* is a null vector with 1 for the 'biomass' reaction flux.

The usual method to solve this linear programming problem is the Simplex algorithm[35], which is fast and scalable. But simplex has no gradient, nor backpropagation compatibility. Thus, it cannot be integrated in AMNs and in any training procedure.

The optimization-RNN proposed by Yang *et al*.[23] is an alternative to solve LP problems, as this method is compatible with backpropagation. The general solving happens by iteratively updating *V* the vector of all reaction fluxes and *M* the dual vector of shadow prices using the following gradients:

$$\nabla V = \left((I - P)[\, c_{FBA} - {S_{iq}}^T R_{iq}] \; - Q R_{int}\right)$$
$$\nabla M = \frac{1}{2}\left(R_{iq} \; - M\right) \qquad (16)$$

where:

$$R_{iq} = \; ReLU\left(M + S_{iq} V - d\right)$$
$$R_{eq} = \; S_{eq} V - b$$
$$Q \; = \; S_{eq}^T \;\; \left(S_{eq}\, S_{eq}^T\right)^{-1}$$
$$P \; = Q\, S_{eq}$$

Further details on this derivation and in particular the form of the matrices S_{eq}, S_{iq} and vectors b, d in the EB and UB cases can be found in the Supplementary Information – section LP-solver equations.

The method converges for both EB and UB cases to the global optimum independently of the initialization. The learning rate was arbitrarily fixed to 0.3 for both EB and UB.

For both Fig. 2d and Fig. 2e the results were obtained with 100 simulations and 10^6 iterations. The reference values were obtained using Cobrapy[22] and the *E. coli* core model[29]. Each replicate was generated with a different medium computed with the training set generation method (see Method section 'Training Set Generation').

AMN architectures and parameters

As listed in Table 1, we propose three AMNs architectures: AMN-Wt, AMN-LP and AMN-QP. The AMNs are run with training sets with exact medium values given at steady state (EB) or only upper bound values (UB). The architecture of RNN-Wt is shown in Fig. 1e and detailed in Supplementary Information – AMN-Wt architecture. The architectures of AMN-LP/QP are shown in Fig. 3a. All ANMs take as input a vector of fluxes of size n_{in} for medium uptake fluxes (V_{in}), then transforms via a dense neural network the input vector into an initial vector of size *n* for all fluxes (V^0), which is refined through an iterative procedure computing $V^{(t+1)}$ from $V^{(t)}$. With all AMNs a n_{in} x n weight matrix transforming V_{in} to V^0 is learned during training, and we name this transforming layer the neural layer. With AMN-LP/QP, $V^{(t)}$

is iteratively updated in a mechanistic layer by the gradient (∇V) of LP/QP solvers (*cf.* previous subsections in Method). With AMN-Wt, the mechanistic layer computes $V^{(t+1)}$ from $V^{(t)}$using the transformations shown in Supplementary Information – Fig. S1, which include a n x m weight matrix (W_r). That weight matrix is learned during training. With all AMN architectures, the values of V corresponding to V_{in} are not updated in the neural or mechanistic layers when training with exact values for medium uptake (EB training sets).

For all architecture, we use the Mean Squared Error (MSE, eq. (2)) on all fluxes as the objective function to minimize while learning. In all AMN architectures we add to the MSE loss function, the terms corresponding to the 3 losses derived from the constraints of the metabolic model (eqs. (3-5)).

The parameters used when training AMNs can be decomposed into two categories:

1. Simulated data parameters. As described in the previous section (Generation of Training Sets with FBA), we can tune the size of the training set to be generated. We can also modify the mean number of selected variable intake medium fluxes, and the number of levels *(i.e.*, the resolution) of the fluxes. We can also modify the actual variable intake medium flux list, but this modifies the architecture of the model (initial layer size), so we kept the same list for each model in the present work.
2. Learning parameters. During learning on simulated data, an AMN has a small set of parameters to tune: the number and size of hidden layers, the number of epochs, the batch size, the dropout ratio and the number of folds in cross-validation.

Optimization of uptake fluxes upper bounds in FBA

The goal of this optimization was to find the best scaler for fluxes to best match experimentally determined growth rates, by using 'out-of-the-box' FBA, simply informing the presence or absence of the flux according to the experimental medium composition. The optimal scaler was found using the Cobrapy software package[22] by simply searching for the maximum R^2 between experimental and FBA-predicted growth rates. The plot showing how the R^2 behaves as a function of the scaler is shown in Fig. 5f.

Data availability

All raw and processed experimental data can be found in our Github repository at https://github.com/brsynth/amn_release/tree/main/Dataset_experimental. Some supplementary files (including 'Data_Fig4.xlsx' and 'Data_Fig5.xlsx') and all other datasets used to produce figures of this study can be also found in our Github repository, at https://github.com/brsynth/amn_release/tree/main/Result.

Code availability

All codes are available within a documented GitHub repository at https://github.com/brsynth/amn_release. The GitHub repository includes tutorials in Google Colab notebooks. All AMN codes make use of Cobrapy[22], numpy[36], scipy[37], Pandas[38], tensorflow[39], sci-kit learn[40] and keras[41] libraries.

References

1. Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583–589 (2021).
2. Bellman, R. *Dynamic Programming*. (Princeton University Press, 1957).
3. Thornburg, Z. R. *et al.* Fundamental behaviors emerge from simulations of a living minimal cell. *Cell* **185**, 345-360.e28 (2022).
4. Reed, J. L. & Palsson, B. Ø. Thirteen years of building constraint-based in silico models of Escherichia coli. *J. Bacteriol.* **185**, 2692–2699 (2003).
5. Beg, Q. K. *et al.* Intracellular crowding defines the mode and sequence of substrate uptake by *Escherichia coli* and constrains its metabolic activity. *Proc. Natl. Acad. Sci. U. S. A.* **104**, 12663–12668 (2007).
6. Goelzer, A. *et al.* Quantitative prediction of genome-wide resource allocation in bacteria. *Metab. Eng.* **32**, 232–243 (2015).
7. Niedenführ, S., Wiechert, W. & Nöh, K. How to measure metabolic fluxes: a taxonomic guide for 13C fluxomics. *Curr. Opin. Biotechnol.* **34**, 82–90 (2015).
8. Willemsen, A. M. *et al.* MetDFBA: incorporating time-resolved metabolomics measurements into dynamic flux balance analysis. *Mol. Biosyst.* **11**, 137–145 (2015).
9. Alghamdi, N. *et al.* A graph neural network model to estimate cell-wise metabolic flux using single-cell RNA-seq data. *Genome Res.* **31**, 1867–1884 (2021).
10. Kim, M., Rai, N., Zorraquino, V. & Tagkopoulos, I. Multi-omics integration accurately predicts cellular state in unexplored conditions for Escherichia coli. *Nat. Commun.* **7**, 13090 (2016).
11. Lewis, J. E. & Kemp, M. L. Integration of machine learning and genome-scale metabolic modeling identifies multi-omics biomarkers for radiation resistance. *Nat. Commun.* **12**, 2700 (2021).

12. Zampieri, G., Vijayakumar, S., Yaneske, E. & Angione, C. Machine and deep learning meet genome-scale metabolic modeling. *PLoS Comput. Biol.* **15**, e1007084 (2019).
13. Sahu, A., Blätke, M.-A., Szymański, J. J. & Töpfer, N. Advances in flux balance analysis by integrating machine learning and mechanism-based models. *Comput. Struct. Biotechnol. J.* **19**, 4626–4640 (2021).
14. Fortelny, N. & Bock, C. Knowledge-primed neural networks enable biologically interpretable deep learning on single-cell sequencing data. *Genome Biol.* **21**, 190 (2020).
15. Lagergren, J. H., Nardini, J. T., Baker, R. E., Simpson, M. J. & Flores, K. B. Biologically-informed neural networks guide mechanistic modeling from sparse experimental data. *PLoS Comput. Biol.* **16**, e1008462 (2020).
16. 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. *J. Comput. Phys.* **378**, 686–707 (2019).
17. SciML: Open Source Software for Scientific Machine Learning. https://sciml.ai/.
18. Müller, S., Regensburger, G. & Steuer, R. Resource allocation in metabolic networks: kinetic optimization and approximations by FBA. *Biochem. Soc. Trans.* **43**, 1195–1200 (2015).
19. Nilsson, A., Peters, J. M., Meimetis, N., Bryson, B. & Lauffenburger, D. A. Artificial neural networks enable genome-scale simulations of intracellular signaling. *Nat. Commun.* **13**, 3069 (2022).
20. Monk, J. M. *et al.* iML1515, a knowledgebase that computes Escherichia coli traits. *Nat. Biotechnol.* **35**, 904–908 (2017).
21. Norsigian, C. J. *et al.* BiGG Models 2020: multi-strain genome-scale models and expansion across the phylogenetic tree. *Nucleic Acids Res.* **48**, D402–D406 (2020).
22. Ebrahim, A., Lerman, J. A., Palsson, B. O. & Hyduke, D. R. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Syst. Biol.* **7**, 74 (2013).
23. Yang, Y., Cao, J., Xu, X., Hu, M. & Gao, Y. A new neural network for solving quadratic programming problems with equality and inequality constraints. *Math. Comput. Simul.* **101**, 103–112 (2014).
24. Jin, L., Li, S., Hu, B. & Liu, M. A survey on projection neural networks and their applications. *Appl. Soft Comput.* **76**, 533–544 (2019).
25. Hopfield, J. J. & Tank, D. W. “Neural” computation of decisions in optimization problems. *Biol. Cybern.* **52**, 141–152 (1985).
26. Wang, J. & Chankong, V. Recurrent neural networks for linear programming: Analysis and design principles. *Comput. Oper. Res.* **19**, 297–311 (1992).
27. Ghasabi-Oskoei, H. & Mahdavi-Amiri, N. An efficient simplified neural network for solving linear and quadratic programming problems. *Appl. Math. Comput.* **175**, 452–464 (2006).
28. Varma, A. & Palsson, B. O. Metabolic capabilities of Escherichia coli: I. synthesis of biosynthetic precursors and cofactors. *J. Theor. Biol.* **165**, 477–502 (1993).
29. Orth, J. D., Fleming, R. M. T. & Palsson, B. Ø. Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide. *EcoSal Plus* **4**, (2010).
30. Cuomo, S. *et al.* Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What’s Next. *J. Sci. Comput.* **92**, 88 (2022).
31. Chubukov, V. *et al.* Transcriptional regulation is insufficient to explain substrate-induced flux changes in Bacillus subtilis. *Mol. Syst. Biol.* **9**, 709 (2013).
32. Tanaka, G. *et al.* Recent advances in physical reservoir computing: A review. *Neural Netw.* **115**, 100–123 (2019).

33. Lachance, J.-C. *et al.* BOFdat: Generating biomass objective functions for genome-scale metabolic models from experimental data. *PLoS Comput. Biol.* **15**, e1006971 (2019).
34. Pandi, A. *et al.* Metabolic perceptrons for neural computing in biological systems. *Nat. Commun.* **10**, 3880 (2019).
35. Karloff, H. The Simplex Algorithm. in *Linear Programming* 23–47 (Birkhäuser Boston, 2009).
36. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).
37. Virtanen, P. *et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* **17**, 261–272 (2020).
38. McKinney. pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing*.
39. Abadi, M. *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv [cs.DC]* (2016).
40. Pedregosa, Varoquaux & Gramfort. Scikit-learn: Machine learning in Python. *the Journal of machine*.
41. Chollet, F. *et al*. Keras. https://keras.io (2015).

Acknowledgements

JLF would like to acknowledge funding provided by the ANR funding agency grant numbers ANR-18-CE44-0015 (SynBioDiag project) and ANR-21-CE45-0021-01 (AMN project) and the BIOS UE HORIZON program (grant number 101070281). LF is supported by INRAE's MICA department and by INRAE's metaprogram DIGIT-BIO. BM is supported by an Ecole Normale Supérieure Scholarship. We thank Aymeric Gaudin (CentraleSupélec Engineering School) for early development in reservoir computing with AMN, Ivan Radkevich (University of Paris Saclay) for his work on custom RNN cells and Tom Lorthios and Hadi Jbara (AgroParisTech and University of Paris Saclay) for their help on collecting data for experimental training sets.

Author contributions

LF and JLF wrote the core of the text of the manuscript. JLF designed the study and wrote the QP solver of Fig. 2 and all the AMN and RC codes used to produce data for Figs. 1, 3, 4, 5 and Table 1. BM wrote the LP-solver producing Figs. 2, S4 and S5 and wrote the corresponding part in the method section. LF benchmarked all codes, wrote the codes transforming SBML models into unidirectional networks and processing experimental data, and handled the colab and git implementations. LF also performed all experimental work reported in Figs. 4 and 5 and wrote the corresponding experimental method section. WL contributed to designing the project and was involved in the discussions and writing the manuscript. All authors read, edited, and approved the manuscript.