A Universal Differential Equation Approach to Tumor Growth Modeling.

Bridging Biological Laws and Neural Dynamics

SINADINOVIC Marko

Abstract

This project applies Universal Differential Equations (UDEs) to decode the unknown dynamics of tumor growth. By embedding a neural network directly into the Gompertz mechanistic model, we aim to capture biological factors that standard equations miss. A key finding of our analysis is the fragility of the popular Adjoint Method when dealing with stiff biological systems. In response, we adopted a high-precision architecture based on Forward-Mode Sensitivity Analysis. By computing exact gradients via Dual Numbers, this approach ensures numerical stability and allows us to isolate the specific mathematical form of the missing dynamics. Experimental benchmarks demonstrate that this hybrid framework outperforms pure Deep Learning by a factor of 57× on unseen dosing protocols, confirming its ability to learn causal biological laws rather than merely overfitting temporal correlations.

Contents

Introduction : From Biological Mechanisms to Hybrid Modeling

Context : The Limits of Theory

When we study population dynamics we learn that growth follows fixed laws. Whether tracking bacteria in a dish or cells in a tissue, the Gompertz or Logistic models seem to provide the perfect answer : a curve that starts slowly, accelerates, and eventually stabilizes as resources deplete. On paper it works perfectly.

However, clinical reality is far more chaotic. A tumor is not just a mathematical curve following a predefined path. It is a complex, adaptive entity that fights back against the immune system and resists treatments in ways that standard equations fail to predict. We find ourselves in a paradoxical situation: we have physical laws that are globally true, but locally insufficient to describe the specific reality of a patient. We are missing a piece of the puzzle.

From Biological Phenomenon to Modeling

To find this missing piece, we must dissect the dynamics. We observe two distinct forces at play. The natural tendency of cells to multiply which follows known physical laws (Gompertz) and the unknown response to chemotherapy or the onset of metabolic resistance.

Instead of choosing between a physical model (too rigid) and a pure neural network (too opaque), we choose to merge them. We retain the ordinary differential equation to guide the general behavior what we know and we inject a Neural Network to learn the missing dynamics what we don't know. By confronting the rigor of physics with the flexibility of Deep Learning we enter the framework of Universal Differential Equations (UDEs).

Seeing to Understand

Now that the concept is defined, the goal is to manipulate it. In advanced mathematics, it is easy to hide behind complex theorems. Here, we take a different path. We want to demystify Scientific Machine Learning through a pragmatic experimental approach.

We will not burden the reader with abstract proofs. Instead, we will define the strict necessary mathematical playground : the limitations of the classic Adjoint Method and the superior stability of Forward-Mode Sensitivity Analysis. The core of this work is a digital petri dish. We will watch the neural network learn in real-time. We want to see, graphically, the exact moment where the machine grasps the hidden biological law and superimposes its prediction onto reality.

1 The Theory of Tumor Growth Modeling

Before introducing complex learning algorithms it is imperative to understand the underlying physical laws we intend to augment. In oncology the growth of a solid tumor is rarely linear. It follows specific dynamics governed by resource availability like oxygen, glucose and spatial constraints.

1.1 The Classical Gompertz Model

Mathematical Formulation and Biological Interpretation

The simplest model for population growth is the Malthusian model, which is a exponential model, where growth is proportional to the current population. But a tumor cannot grow indefinitely, it is limited by the carrying capacity of its host tissue.

To account for this saturation Benjamin Gompertz proposed a model in 1825 where the growth rate decays exponentially over time. The dynamics of the tumor volume $N(t)(t)$ are governed by the following Ordinary Differential Equation (ODE) :

$$(\Psi) : \begin{cases} \frac{dN}{dt} = f(t, N(t)) = rN(t) \ln\left(\frac{K}{N(t)}\right) \\ N(0) = N_0 \end{cases} \tag{1}$$

Where:

- $N(t) \in \mathbb{R}_+^*$ is the tumor size (volume or number of cells) at time t.
- $r > 0$ represents the intrinsic proliferation rate.
- $K > 0$ is the carrying capacity (the maximum size the tumor can reach).

Biologically, the term $\ln(K/N(t))$ acts as a brake. When $N(t)$ is small ($N(t) \ll K$), the term is large, and growth is rapid. As $N(t)$ approaches K, the term tends to 0, and growth halts. Indeed $\ln(K/N(t)) = \ln(K) - \ln(N(t))$.

Analytical Resolution and Stability Analysis

To legitimize the use of this model in a computational framework we must first ensure the problem is well-posed in the sense of Hadamard. It must satisfies three criteria :

- **Existence :** We must ensure that our differential equations actually yield a solution that aligns with the biological parameters we observe
- **Uniqueness :** It is vital that our model produces a single and clear trajectory
- **Stability :** The system must be resilient to small changes. In the real world, biological data is often noisy, so we need to be sure that slight experimental errors won't lead to divergent results

a) Existence and uniqueness We invoke the fundamental theorem of ODEs.

(Theorem) Cauchy-Lipschitz Theorem - Local Version
Consider the Cauchy Problem :

$$\begin{cases} \frac{dy}{dt} = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

Let $\mathcal{O}$ be an open set in $\mathbb{R} \times \mathbb{K}^n$, and let $f : \mathcal{O} \to \mathbb{K}^n$ If f is continuous in t and locally Lipschitz continuous with respect to the state variable y, given any $(t_0, y_0) \in \mathcal{O}$, there exists some $\alpha > 0$ such that the Cauchy Problem $y(t_0) = y_0$ for (Ψ) possesses a unique solution y defined on the interval $[t_0 - \alpha, t_0 + \alpha]$.

Let $f(N) = rN(\ln K - \ln N) \in C^1(0, +\infty)$. Here N represents the state variable (tumor size). To check the Lipschitz condition we differentiate f with respect to the state variable N:

$$\frac{df}{dN} = r \ln(K/N) + rN\left(-\frac{1}{N}\right) = r(\ln(K/N) - 1)$$

Since f is continuously differentiable on the open domain $\mathbb{R}_+^*$ it is locally Lipschitz. Thus, for any initial condition $N_0 > 0$, there exists a unique solution. Since equation (Ψ) is a separable ODE, we can solve it explicitly.

b) Analytical Solution

Let us introduce the change of variable $u(t) = \ln(N(t)/K)$ and by differentiating with respect to time :

$$\frac{du}{dt} = \frac{d}{dt}(\ln N(t) - \ln K) = \frac{1}{N(t)}\frac{dN}{dt}$$

Substituting $\frac{dN}{dt}$ from the Gompertz equation:

$$\frac{du}{dt} = \frac{1}{N(t)}\left(rN(t)\ln\left(\frac{K}{N(t)}\right)\right) = r\ln\left(\frac{K}{N(t)}\right)$$

Noting that $\ln(K/N) = -\ln(N/K) = -u(t)$, this reduces the nonlinear Gompertz equation to a simple linear decay equation:

$$\frac{du}{dt} = -ru(t) \implies u(t) = u(0)e^{-rt}$$

Substituting back $N(t) = Ke^{u(t)}$, we obtain the closed-form solution:

$$N(t) = K\exp\left(\ln\left(\frac{N_0}{K}\right)e^{-rt}\right) \tag{2}$$

c) Fixed Points and Stability Analysis

Now we search for equilibrium states N^* (fixed points) such that : $f(N^*) = 0$ (the growth rate vanishes)

$$rN^*\ln\left(\frac{K}{N^*}\right) = 0$$

This equation yields two solutions:

- A Trivial equilibrium : $N_0^* = 0$ (extinction).
- $\ln(K/N^*) = 0 \implies N^* = K$ (Carrying capacity).

To determine the local stability of the non-trivial equilibrium $N^* = K$, we evaluate the sign of the derivative of the growth function with respect to the state variable N at the point K :

$$\frac{df}{dN} = \frac{d}{dN}[rN(\ln K - \ln N)] = r(\ln(K/N) - 1)$$

Evaluating at $N = K$:

$$\left.\frac{df}{dN}\right|_{N=K} = r\left(\ln(1) - 1\right) = -r$$

Since $r > 0$, the derivative is negative $(-r < 0)$.

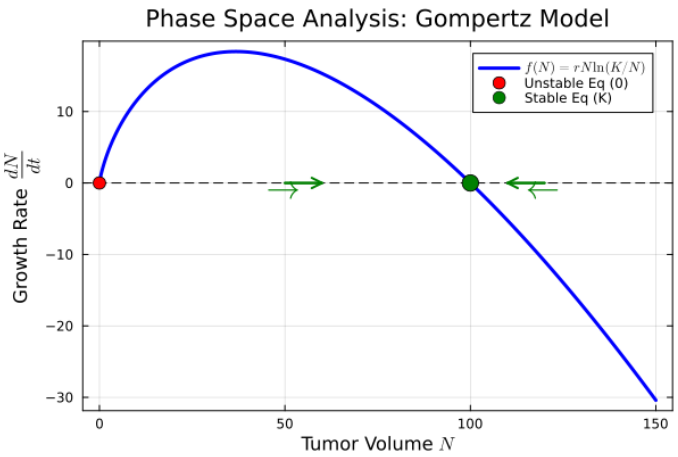

Figure 1:

To conclude rigorously we invoke a fundamental result of dynamical systems theory.

(Theorem) Lyapunov's Indirect Method / Linearization Principle Theorem
Let $\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$ be an autonomous dynamical system where $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable, $f \in C^1(\mathbb{R}^n)$. Let $\mathbf{x}^*$ be an equilibrium point $(f(\mathbf{x}^*) = 0)$.

Define $J = Df(\mathbf{x}^*)$ as the Jacobian matrix of f evaluated at the equilibrium:

$$J_{ij} = \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}^*}$$

Let $\sigma(J)$ be the spectrum (set of eigenvalues) of J.

- If for all eigenvalues $\lambda \in \sigma(J)$, $\mathrm{Re}(\lambda) < 0$, then $\mathbf{x}^*$ is locally asymptotically stable, the equilibrium is a sink.
- If there exists at least one eigenvalue $\lambda \in \sigma(J)$ such that $\mathrm{Re}(\lambda) > 0$, then $\mathbf{x}^*$ is unstable (source or saddle).

Let's apply to the scalar case now $(n = 1)$. In our one-dimensional Gompertz model, the Jacobian matrix reduces to the scalar derivative $J = f'(N^*)$. The single eigenvalue is $\lambda = f'(N^*)$. Thus the stability condition $\mathrm{Re}(\lambda) < 0$ simplifies directly to:

$$\left.\frac{df}{dN}\right|_{N^*} < 0$$

Since we calculated $\frac{df}{dN}(K) = -r$ and $r \in \mathbb{R}_+^*$, the eigenvalue is strictly negative, proving asymptotic stability.

It means any trajectory starting close to K will converge towards K and near $N = 0$ the growth rate is positive making 0 an unstable equilibrium. Said differently the tumor will naturally converge towards its carrying capacity K matching clinical observations of untreated tumors.

Limitations : Structural Bias and Missing Dynamics

While the Gompertz model provides a robust baseline for natural growth, relying on it exclusively creates a significant structural bias. By definition the equation (Ψ) assumes that the derivative $\frac{dN}{dt}$ is fully explained by the current state N and two constant parameters r and K.

In a clinical context this assumption collapses. A patient undergoing treatment is subject to external perturbations that the original equation cannot see. Let us assume the true dynamics of the tumor follow a perturbed law:

$$\frac{dN(t)}{dt} = \underbrace{rN(t)\ln\left(\frac{K}{N(t)}\right)}_{\text{Known Physics } (f)} + \underbrace{U(t, N(t))}_{\text{Unknown Physics}} \tag{3}$$

Here $U(t, N)$ represents the missing dynamics. It could model:

- A chemotherapy term $-C(t)N$, where $C(t)$ is the drug concentration.
- An immune response term that saturates over time.
- An acquired resistance mechanism where the efficacy of the drug decays.

If we force a pure Gompertz model ($U = 0$) onto clinical data generated by a system where $U \neq 0$, the optimizer will attempt to compensate for the missing term by distorting the values of r and K. We would obtain "optimal" parameters that are mathematically minimizing the error but biologically false.

We are therefore left with an inverse problem where the functional form of U is unknown. We cannot simply guess it. This motivates the central idea replacing the specific unknown function $U(t, N)$ with a Universal Approximator, a Neural Network, capable of learning the shape of the perturbation directly from the trajectory.

1.2 From ODEs to Universal Differential Equations (UDEs)

The Gray-Box Philosophy

We have established that the classical Gompertz equation is insufficient to capture complex clinical scenarios due to the unknown term $U(t, N)$. This leaves us with a dilemma. We cannot stick to a pure physical model (White-Box) because it is too rigid but we do not want to switch to a pure Deep Learning model (Black-Box) because we would lose the biological interpretability of parameters r and K.

We need a mathematical object capable of respecting the known physics while mimicking any continuous function to fill the gap. This is the essence of Gray-Box modeling.

To implement this philosophy we rely on the framework of Universal Differential Equations (UDEs) popularized by Rackauckas et al. (2020). The core idea is to embed a neural network directly inside the differential equation structure.

We define our hybrid system as follows :

$$\frac{dN(t)}{dt} = \underbrace{f_{phys}(N(t), p)}_{\text{Known Physics}} + \underbrace{\mathcal{NN}_\theta(N(t))}_{\text{Unknown Physics}} \tag{4}$$

Where:

- $f_{phys}(N,p) = rN\ln(K/N)$ the trusted Gompertz growth with physical parameters $p = \{r, K\}$.
- $\mathcal{NN}_\theta : \mathbb{R} \to \mathbb{R}$ is a neural network parametrized by a vector of weights and biases $\theta \in \mathbb{R}^d$ $card(d) < \infty$

It is important to understand that the neural network isn't predicting the future state $N(t+\Delta t)$ directly as a standard Recurrent Neural Network would. Instead it acts as a constituent of the vector field.

At each time step t the numerical solver evaluates the physical term passes the current state $N(t)$ through the neural network to compute the correction term then sums them up to get the total derivative $\frac{dN}{dt}$ and only then integrates this sum to move to the next step. This ensures that the learning process is constrained by the differential structure of the problem.

The problem shifts from finding a function U to finding the optimal parameters θ that minimize the distance between the solution of equation (4) and the observed data.

A legitimate question arises. Why do we choose a Neural Network to model the unknown term $U(N)$? Why not a polynomial or a Fourier series like usual ? The choice is not arbitrary. It relies on a strong result from functional analysis : the Universal Approximation Theorem (UAT). While we have no idea what the mathematical formula for the drug resistance or immune response looks like, we can safely assume it is a continuous process. The UAT gives us the mathematical guarantee that a neural network can mimic any continuous function, no matter how complex, provided it has enough neurons.

(Theorem) Universal Approximation Theorem - Cybenko and Hornik, 1991
Let $\sigma(\cdot)$ be a non-constant, bounded, and continuous activation function (e.g., sigmoid or tanh). Let $K \subset \mathbb{R}^n$ be a compact set. The space of continuous functions implemented by a single-hidden-layer neural network:

$$\mathcal{F} = \left\{ F(x) = \sum_{i=1}^{m} w_i \sigma(a_i^T x + b_i) \mid m \in \mathbb{N}, w_i, b_i \in \mathbb{R} \right\}$$

is dense in $C(K, \mathbb{R})$ with respect to the uniform norm.

In simpler terms, for any continuous function $f : K \to \mathbb{R}$ and any error tolerance $\varepsilon > 0$, there exists a network parameters configuration θ (with sufficient width m) such that:

$$\sup_{x \in K} |f(x) - \mathcal{NN}_\theta(x)| < \varepsilon$$

This theorem is the mathematical foundation of our project. It validates the form of equation (4).

$$\frac{dN(t)}{dt} = \underbrace{rN(t)\ln\left(\frac{K}{N(t)}\right)}_{\text{Known Physics } (f)} + \underbrace{\mathcal{NN}_\theta(N(t))}_{\approx U(N(t))} = \frac{dN}{dt} = \text{Gompertz}(N) + \underbrace{\mathcal{NN}_\theta(N)}_{\approx U(N)} \tag{5}$$

By invoking this theorem, we operate a fundamental shift in our solving strategy. Instead of searching for an unknown function in an infinite-dimensional space, we reduce the problem to finding a vector of parameters θ in $\mathbb{R}^d$. Also we no longer need to guess the equation, just need to find the optimal θ that minimizes the error.

However this theorem is an existence result not a constructive one. It tells us that a good θ exists but it does not tell us how to find it. Finding this optimal θ requires training the network inside the differential equation but it creates a new danger. The network could potentially learn the wrong thing. Before finding

the right parameters we must ensure that our hybrid system is well-posed before calculations.

Structural Identifiability

We have built a system that adds a physical model and a neural network.

$$\frac{dN}{dt} = \text{Gompertz}(N) + \mathcal{NN}_\theta(N)$$

The question is : how do we know who is doing the work ?

Imagine we overestimate the proliferation rate r. The Gompertz term will predict a tumor that grows way too fast. The neural network, whose only goal is to minimize the error, will naturally learn a negative correction just to slow down the equation and match the data. Visually the final curve $N(t)$ will look perfect but scientifically it's a disaster. We would have a false physical parameter compensated by a fake neural term.

If we try to learn parameters r, K and weights θ simultaneously the problem is illposed. There is an infinite number of ways to combine a wrong physical model with a compensating neural network to fit the same dataset:

$$\text{Observed Data} \approx \underbrace{f_{phys}(p_{wrong})}_{\text{Too fast}} + \underbrace{\mathcal{NN}_{compensating}}_{\text{Brakes}}$$

In this scenario the discovery is just a mathematical illusion. The network isn't learning biology it's fixing our mistakes. To counter this we need to discipline the model. We cannot let the neural network touch the parts of the growth that we already understand. We therefore adopt a strict protocol mimicking a clinical trial :

1. We train only the Gompertz parameters (r, K) on untreated tumors. Here the neural network is turned off ($U = 0$). This anchors the white-box model on solid ground.
2. We introduce the data from treated patients. Freeze r and K. The physical model is no longer allowed to change. Consequently any deviation from the natural growth must be captured by the neural network.

By structurally separating these steps we force the network to learn exactly the missing biological mechanism and nothing else.

But to find the optimal parameters θ we need to minimize the error between our prediction and the data. This requires calculating the gradient of the loss function with respect to the neural network weights. But how do we calculate a derivative through a simulation without exploding our computer's memory ? This question leads us to the implementation of the Adjoint Sensitivity Method.

2 The Adjoint Sensitivity Method

Before detailing our specific implementation, it is necessary to establish the mathematical framework of the Adjoint Sensitivity Method. Although we will later demonstrate why a Forward-Mode approach is superior for our specific low-parameter constraint (see Section 3), the Adjoint method remains the theoretical cornerstone of the Neural ODE field and serves as the primary benchmark for our work.

2.1 Formulation of the Inverse Problem

The Loss Function

In a real clinical setting we don't have a continuous video of the tumor growing. We only have snapshots taken during medical exams. For this reason we define our dataset $\mathcal{A}$ as a collection of M discrete observations:

$$\mathcal{A} = \{(t_i, N_i^{obs})\}_{i=1}^{M}$$

Where:

- M is the total number of measurements available e.g. if the patient came for a scan 10 times, $M = 10$
- t_i represents the specific date of the i-th exam e.g. $t_1 = \text{Day } 0, t_2 = \text{Day } 7 \ldots$
- N_i^{obs} is the observed tumor size at that specific time

Our goal is to find the neural network parameters θ such that our model's simulated trajectory, denoted $N(t, \theta)$, passes as close as possible to these observed points. That's why we define the Loss Function $J(\theta)$ as the Mean Squared Error (MSE) :

$$J(\theta) = \frac{1}{M} \sum_{i=1}^{M} \left\| N(t_i, \theta) - N_i^{obs} \right\|^2 \tag{6}$$

We notice the term $N(t_i, \theta)$ is the result of an Cauchy Problem :

$$\begin{cases} \frac{dN}{dt} = f(N, \theta) \\ N(t_0) = N_0 \end{cases}$$

This implies that every time we want to evaluate the error for a specific θ we have to simulate the entire patient's history from t_0 to t_{end}.

Regularization

If we let the neural network minimize $J(\theta)$ without constraints it tends to hallucinate. It might use massive weights to fit the noise in the data creating a wiggly curve that makes no biological sense.

To prevent this we apply Weight Decay. This consists to add a penalty term to the loss function to force the weights θ to remain small. So the total loss becomes :

$$Loss(\theta) = J(\theta) + \alpha \|\theta\|_2^2 \tag{7}$$

Where α is a hyperparameter controlling the strength of the penalty, usually small, e.g. 10^{-4}.

2.2 Derivation of the Continuous Adjoint State

To perform gradient descent we need the sensitivity of the objective functional $J(\theta)$ with respect to the parameters. Since the state $N(t)$ is constrained by a differential equation we cannot differentiate J directly. Instead we employ the Lagrangian formalism adapted to Banach spaces treating the ODE as a constraint in the space of square-integrable functions.

The Lagrangian Formalism and Constrained Optimization

First we establish the functional framework. We define the state trajectory N in the space of continuously differentiable functions $\mathcal{C} = C^1([t_0, t_f], \mathbb{R})$ and the parameter vector θ in $\mathbb{R}^d$.

a) The Constrained Minimization Problem

Our optimization problem is formally stated as minimizing the functional J on the manifold defined by the differential equation. We define the running cost density $L : [t_0, t_f] \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ as the instantaneous measure of the discrepancy between the simulation and the reality.

In our clinical setting observations are sparse. We possess a finite set of measurements $\mathcal{A} = \{(t_i, N_i^{obs})\}_{i=1}^M$. To incorporate these discrete points into the continuous calculus of variations framework, we represent the loss density L using Dirac delta distributions :

$$L(t, N(t), \theta) = \sum_{i=1}^{M} \frac{1}{2\sigma^2} \|N(t) - N_i^{obs}\|^2 \delta(t - t_i) \tag{8}$$

Where σ^2 represents the measurement noise variance and $\delta(\cdot)$ is the Dirac distribution. It bridges the gap between statistics and calculus. When we integrate this density over time, the property of the Dirac distribution $\int f(t)\delta(t - t_i)dt = f(t_i)$ recovers exactly the Sum of Squared Errors (SSE) :

$$\int_{t_0}^{t_f} L(t, N, \theta)dt = \sum_{i=1}^{M} \frac{1}{2\sigma^2} \|N(t_i) - N_i^{obs}\|^2$$

But mathematically minimizing $\sum(x - y)^2$ or minimizing $\sum \frac{1}{\sigma^2}(x - y)^2$ gives exactly the same optimal parameters θ because σ is just a multiplicative constant. Therefore we set the constant $\sigma = 1$:

$$J(N, \theta) = \int_{t_0}^{t_f} L(t, N, \theta)\, dt = \sum_{i=1}^{M} \frac{1}{2} \|N(t_i) - N_i^{obs}\|^2$$

The optimization problem is then formally written as :

$$\min_{\theta \in \mathbb{R}^d} \quad J(N, \theta) = \int_{t_0}^{t_f} L(t, N(t), \theta)\, dt \tag{9}$$

$$\text{subject to} \quad \frac{dN}{dt} - f(t, N, \theta) = 0, \quad \forall t \in [t_0, t_f] \tag{10}$$

b) Geometry of the Constraint

How do we strictly enforce a constraint that must hold at every single instant $t \in [t_0, t_f]$? In finite-dimensional optimization ($\mathbb{R}^n$) the classic Lagrange Multiplier Theorem allows us to adjoin constraints using a vector $\lambda \in \mathbb{R}^k$. Here our constraint is not a set of equations but a differential operator acting on a function space.

However we have a theorem that allows us to use $\lambda \in \mathbb{R}^k$. It's the Lagrange Multiplier Theorem for Banach Spaces. This theorem states that if we minimize a functional subject to an equality constraint $G(N) = 0$ where G maps to a Banach space Z, there exists a continuous linear functional $\lambda^* \in Z^*$ (the topological dual space) such that the Lagrangian is stationary at the optimum.

Here our constraint is not a single point but a continuous trajectory. To handle this rigorously we must extend the concept of the dot product to function spaces.

Let us define the residual of the differential equation as a function $e(N, \theta)$ living in the Hilbert space $\mathcal{H}$ of square-integrable functions $L^2([t_0, t_f])$ and our goal is to force this entire function to be the zero function :

$$e(N, \theta)(t) = \frac{dN}{dt} - f(t, N, \theta) = 0$$

To incorporate this functional constraint into our scalar objective J, we introduce a Lagrange multiplier λ. Mathematically λ acts as a linear functional in the topological dual space of L^2. We can represent this action using the canonical L^2 inner product. For any two real-valued functions u, v defined on $[t_0, t_f]$:

$$\langle u, v \rangle_{L^2} = \int_{t_0}^{t_f} u(t)v(t)\, dt \tag{11}$$

Think of this as the continuous limit of the standard dot product. Just as $\lambda^T x = \sum \lambda_i x_i$ for vectors the pairing in function space becomes an integral:

$$\langle \lambda, e(N, \theta) \rangle_{L^2} = \int_{t_0}^{t_f} \lambda(t) \underbrace{\left(\frac{dN}{dt} - f(t, N, \theta) \right)}_{\text{Constraint error}} dt \tag{12}$$

This defines $\lambda(t)$ not just as a variable but as a time-dependent function : the Adjoint State. It weights the importance of the constraint violation at each moment in time.

c) The Lagrangian Functional

Having defined the scalar product structure we can now construct the central object of our optimization method. We define the Lagrangian $\mathcal{L} : \mathcal{C} \times \mathbb{R}^d \times \mathcal{C}^* \to \mathbb{R}$. It's constructed by subtracting the inner product of the multiplier and the constraint from the objective functional:

$$\mathcal{L}(N, \theta, \lambda) = J(N, \theta) - \left\langle \lambda, \frac{dN}{dt} - f(N, \theta, t) \right\rangle_{L^2} \tag{13}$$

It can be rewritten in its integral form as follows :

$$\mathcal{L}(N, \theta, \lambda) = \int_{t_0}^{t_f} L(N, \theta, t)\, dt - \int_{t_0}^{t_f} \lambda(t) \left(\frac{dN}{dt} - f(N, \theta, t) \right) dt \tag{14}$$

This functional combines the objective we want to minimize (J) and the physics we must respect.

(Proposition) Consistency on the Feasible Manifold

Let $\mathcal{M}$ be the feasible manifold defined by the set of pairs (N, θ) that strictly satisfy the ODE constraint. For any feasible pair $(N, \theta) \in \mathcal{M}$, the residual term vanishes identically. Consequently :

$$\mathcal{L}(N, \theta, \lambda) = J(N, \theta), \quad \forall \lambda \in L^2([t_0, t_f])$$

This identity justifies the method of Lagrange Multipliers. It implies that the constrained extrema of the original functional J correspond to the stationary points of the unconstrained Lagrangian $\mathcal{L}$.

Instead of minimizing J directly which is difficult because of the complex constraint, our strategy is to find the triplet (N, θ, λ) that makes the Lagrangian stationary. Mathematically we require the total variation of $\mathcal{L}$ to vanish:

$$\delta\mathcal{L} = 0 \implies \begin{cases} \delta_N \mathcal{L} = 0 & \text{(Adjoint Equation)} \\ \delta_\lambda \mathcal{L} = 0 & \text{(State Equation)} \\ \delta_\theta \mathcal{L} = 0 & \text{(Gradient Condition)} \end{cases}$$

2.3 Derivation of the Adjoint Equation

We now calculate the variation of the Lagrangian $\mathcal{L}$ with respect to the state variable N. We consider a perturbation $\delta N \in \mathcal{C}$ around the optimal trajectory. For $\mathcal{L}$ to be stationary with respect to state variations,we require the Fréchet derivative $\delta_N \mathcal{L}$ to vanish for all arbitrary perturbations δN.

a) Total Variation Calculation

Let us perturb the trajectory $N(t) \to N(t) + \delta N(t)$ while keeping θ and λ fixed. The linear variation of $\mathcal{L}$ is given by:

$$\delta_N \mathcal{L}(\delta N) = \int_{t_0}^{t_f} \frac{\partial L}{\partial N} \delta N \, dt - \int_{t_0}^{t_f} \lambda(t) \left(\frac{d}{dt}(\delta N) - \frac{\partial f}{\partial N} \delta N \right) dt \tag{15}$$

Rearranging the terms to group those multiplying δN and those involving the derivative $\frac{d}{dt}(\delta N)$:

$$\delta_N \mathcal{L} = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial N} + \lambda^T \frac{\partial f}{\partial N} \right) \delta N \, dt - \int_{t_0}^{t_f} \lambda^T \frac{d}{dt}(\delta N) \, dt \tag{16}$$

b) Integration by Parts

The term $\frac{d}{dt}(\delta N)$ is problematic because it prevents us from factoring out δN. To resolve this we apply Integration by Parts to transfer the time derivative from the perturbation δN to the adjoint state λ.

Since $(\lambda, N) \in \mathcal{C}^2$:

$$\int_{t_0}^{t_f} \lambda(t) \frac{d}{dt}(\delta N) \, dt = [\lambda(t) \delta N(t)]_{t_0}^{t_f} - \int_{t_0}^{t_f} \frac{d\lambda}{dt} \delta N(t) \, dt \tag{17}$$

Substituting this result back into (16) our variation becomes :

$$\delta_N \mathcal{L} = \int_{t_0}^{t_f} \underbrace{\left(\frac{\partial L}{\partial N} + \lambda^T \frac{\partial f}{\partial N} + \frac{d\lambda^T}{dt} \right)}_{\text{Must be zero}} \delta N \, dt - \left[\lambda^T \delta N \right]_{t_0}^{t_f} \tag{18}$$

c) The Adjoint Differential Equation

For $\mathcal{L}$ to be a stationary point, the integral term must vanish for any arbitrary perturbation δN. This implies :

$$\frac{d\lambda^T}{dt} + \lambda^T \frac{\partial f}{\partial N} + \frac{\partial L}{\partial N} = 0$$

By taking the transpose of this entire equation to return to the standard column-vector notation where $\nabla L = (\frac{\partial L}{\partial N})^T$ we obtain the final Adjoint Equation :

$$\frac{d\lambda}{dt} = -\left(\frac{\partial f}{\partial N}\right)^T \lambda - \nabla_N L \tag{19}$$

This result describes how the sensitivity vector $\lambda(t)$ evolves over time. Mathematically strictly note that even if the original physical system f is highly non-linear like Gompertz growth, the adjoint equation is always a Linear Time-Varying system with respect to λ.

The presence of the transposed Jacobian $\left(\frac{\partial f}{\partial N}\right)^T$ is the signature of a dual problem. While the physical state N flows forward along the vector field the adjoint state λ flows upstream against it gathering information about the gradient.

However a linear ODE requires a boundary condition to be solvable. Since the equation depends on the derivative of the loss at time t its anchor naturally lies in the future, at $t = t_f$. This leads us to the boundary conditions derived in the next section.

The Transversality Condition

We are left with the boundary term from the integration by parts:

$$B = -\left(\lambda(t_f)^T \delta N(t_f) - \lambda(t_0)^T \delta N(t_0)\right) \tag{20}$$

We analyze the physical constraints on the endpoints :

- $t = t_0$ **:** The initial condition of the tumor is fixed $N(t_0) = N_0$. Therefore no variation is allowed $\delta N(t_0) = 0$ and the term $\lambda(t_0)^T \delta N(t_0)$ vanishes naturally.
- $t = t_f$ **:** The final state do not impose a target tumor size. Thus $\delta N(t_f)$ is arbitrary. For the boundary term to vanish for any $\delta N(t_f)$ we must impose $\lambda(t_f) = 0$

To compute the gradien we must first solve the state equation forward from t_0 to t_f and then solve the adjoint equation backward in time from t_f to t_0 using the final condition $\lambda(t_f) = 0$.

2.4 Gradient Computation

We have derived the state dynamics (forward) and the adjoint dynamics (backward). The final piece of the puzzle is to compute the gradient of the objective function J with respect to the parameters θ. This gradient is what feeds the optimizer (e.g., ADAM or BFGS) to update the weights.

Integral Form of the Gradient

We return to the total variation of the Lagrangian. The condition for stationarity with respect to parameters is $\delta_\theta \mathcal{L} = 0$. Differentiation of the Lagrangian functional with respect to θ gives :

$$\nabla_\theta J = \nabla_\theta \mathcal{L} = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial \theta} + \lambda(t)^T \frac{\partial f}{\partial \theta}\right) dt \tag{21}$$

In our specific case the vector field is defined as $f(N, \theta) = \text{Gompertz}(N) + \mathcal{NN}_\theta(N)$. The mechanistic part (Gompertz) does not depend on the neural network weights θ. Therefore the Jacobian reduces to the neural network's contribution :

$$\frac{\partial f}{\partial \theta} = \frac{\partial \mathcal{NN}_\theta(N)}{\partial \theta}$$

Assuming the regularization is explicit outside the integral, e.g. $+\alpha\|\theta\|^2$, the final gradient formula is :

$$\frac{dJ}{d\theta} = \int_{t_0}^{t_f} \lambda(t)^T \frac{\partial \mathcal{NN}_\theta(N(t))}{\partial \theta}\, dt + 2\alpha\theta \tag{22}$$

This integral has a clear physical meaning. The term $\frac{\partial \mathcal{NN}}{\partial \theta}$ represents how the network's output changes locally when weights are tweaked. The adjoint state $\lambda(t)$ acts as a time-dependent weight, it amplifies the gradient at times where the model makes large errors i.e. where sensitivity is high and suppresses it where the model is already accurate.

3 Computational Implementation in Julia

3.1 Complexity Analysis

When training a neural network embedded in a differential equation, the choice of the gradient computation method is not merely a technical detail but a fundamental trade-off between memory efficiency and numerical stability. This decision depends on two competing dimensions:

1. **The Parameter Space :** P the number of trainable parameters in the network.
2. **Numerical Stability :** The requirement for exact gradient computation which is crucial for scientific discovery where we aim to identify physical laws rather than just fitting curves.

The following table summarizes why, contrary to standard Deep Learning trends, we diverged from the Adjoint method for this specific high-fidelity simulation:

Method	Complexity & Constraints	Suitability for this Project
1. Forward-Mode Sensitivity	**Complexity:** $O(P)$ Calculates exact derivatives by extending the state vector with the Jacobian. Highly stable and precise.	**Selected Strategy.** Since our optimized anti-stiffness architecture is compact ($P \approx 300$) this method offers the highest numerical precision without prohibitive computational cost.
2. Discrete Backprop (BPTT)	**Memory:** $O(T \times N_{state})$. Requires storing the state at *every* step of the solver (treating the solver as a giant RNN).	**Infeasible.** For high-fidelity explicit solvers like `Vern7` ($T > 10^4$) this approach hits the memory wall crashing GPU/CPU RAM.
3. Adjoint Method	**Memory:** $O(1)$ (Constant). Solves the ODE backward in time to reconstruct gradients.	**Rejected.** It's memory efficiency but reverse-mode integration often suffers from numerical drift masking the true physical dynamics of biological systems.

Table 1: Comparison of gradient computation strategies.

Multiple Dispatch and the Composability Advantage

A recurring hurdle in scientific computing is the Two-Language Problem, where models are prototyped in a high-level language but rewritten in C or Fortran for execution. Julia bypasses this by using Multiple Dispatch a paradigm that allows the compiler to generate specialized machine code based on the types of all function arguments.

In the context of our tumor growth model this results in unparalleled composability :

- **Type-Agnostic Solvers :** The ODE solvers in `DifferentialEquations.jl` do not know they are integrating a neural network. They are written to handle any callable object that satisfies the derivative interface.
- **Vertical Integration :** Because the neural networks via `Lux.jl` and the solvers are both written in native Julia automatic differentiation tools like `Zygote.jl` can differentiate through the entire solver's logic.

3.2 Sensitivity Analysis

The bridge between our numerical solver and the gradient-based optimizer is the sensitivity analysis. This choice determines how the gradients of the loss function with respect to the parameters θ are propagated through the temporal integration of the tumor dynamics.

a) Continuous Adjoint (The Standard Approach)

This framework treats the ODE as a continuous functional object, deriving an Adjoint ODE to propagate gradients backward in time. While highly memory efficient because it avoids storing intermediate solver stages the resulting gradient corresponds to the ideal mathematical ODE rather than the specific numerical approximation. In stiff biological systems this adjoint mismatch can destabilize the training process leading to drifting gradients.

b) Forward-Mode Sensitivity (Our Approach)

Unlike standard backpropagation which traverses a computational graph backward, this approach computes the sensitivities $\frac{du}{d\theta}$ simultaneously with the solution $u(t)$. It leverages dual numbers, an algebraic trick where every number carries its own derivative e.g. $x + \epsilon x'$. As the solver advances from t to $t + dt$ it automatically updates both the state of the tumor and the exact derivative of that state with respect to the neural weights.

3.3 Conclusion on Implementation Strategy

While the Adjoint method is the industry standard for large-scale Deep Learning e.g.Computer Vision with millions of parameters, our focus prioritizes numerical exactness.

Consequently we adopt Forward-Mode Sensitivity Analysis via `ForwardDiff.jl`. This approach allows us to compute exact gradients for our compact architecture ensuring that the optimization landscape is not corrupted by the approximation errors inherent to reverse time integration. This choice effectively transforms our UDE into a robust solver capable of uncovering the hidden trap dose dynamics.

This means that when we compute the gradient of our loss function we aren't just treating the ODE solver as a black box external library. The compiler optimizes the interaction between the neural network's weights θ and the solver's internal stages k_i as a single unified mathematical operation.

Having established the theoretical framework for our optimization strategy, we now transition from the mathematical "why" to the structural "how" detailing the implementation of these dynamics.

4 Software Architecture

Implementing a Universal Differential Equation requires a tight integration between deep learning primitives and numerical analysis tools. We rely on the `SciML` (Scientific Machine Learning) ecosystem in Julia specifically `DifferentialEquations.jl` for the solver engine and `Lux.jl` for the neural parameterization.

4.1 ODE Solvers

he numerical integrator serves as the core component of our training architecture. To understand the efficiency of the algorithms within `DifferentialEquations.jl` it is essential to revisit the mathematical foundation of ODE integration: the Runge-Kutta family.

Explicit Runge-Kutta Methods

Numerical integration aims to solve $\frac{dy}{dt} = f(t, y)$ by stepping forward in time ($y_{n+1} = y_n + \Delta y$). The classical pedagogical approach is the Runge-Kutta 4 (RK4) method. It consists of taking four samples of the derivative to construct a weighted average. For a step size h:

$$k_1 = f(t_n, y_n) \qquad \text{(Derivative at the beginning)} \tag{23}$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \qquad \text{(First estimate at midpoint)} \tag{24}$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \qquad \text{(Second estimate at midpoint)} \tag{25}$$

$$k_4 = f(t_n + h, y_n + hk_3) \qquad \text{(Estimate at the end)} \tag{26}$$

The next state is computed by a specific weighted sum known as Simpson's quadrature:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \tag{27}$$

Although RK4 is robust for simple problems, it fails when applied to stiff equations systems where variables evolve at drastically different speeds. In a UDE the neural network $\mathcal{NN}_\theta(x)$ can initially produce high frequency oscillations. If we applied a fixed step solver like RK4 we would be forced to reduce h to infinitesimal values ($h < 10^{-6}$) to prevent the solution from exploding making training computationally intractable.

Adaptive Step Sizing and Embedded Methods

To introduce numerical safety and efficiency we transition to Adaptive Step Sizing. Modern solvers do not use a single method but a pair of methods embedded in one another.

Formalized by the Butcher Tableau (Eq. 28) these methods compute two approximations of the next state using the same intermediate slopes k_i but with different weighting vectors $\mathbf{b}$ (high order) and $\hat{\mathbf{b}}$ (lower order) :

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \quad \text{(Solution)} \\ & \hat{\mathbf{b}}^T \quad \text{(Error Estimator)} \end{array} \tag{28}$$

This allows the solver to estimate the local error $\epsilon = \|y_{n+1} - \hat{y}_{n+1}\|$ at virtually zero cost. The step size h is

then dynamically adjusted :

$$h_{new} = h \cdot \left(\frac{\text{Tolerance}}{\epsilon} \right)^{\frac{1}{\text{Order}+1}}$$

Verner's Method

For our specific goal of scientific discovery requires higher precision. We need to distinguish subtle biological signals like the drug interaction from numerical noise. We adopted the Verner's 7/6 Runge-Kutta method denoted `Vern7`.

Property	**Specification**
Algorithm	Verner's 7/6 (`Vern7`)
Numerical Order	7^{th} Order (Solution)
Error Estimator	6^{th} Order (Adaptive Step Control)
Target Tolerance	High Fidelity ($\approx 10^{-10}$)
Classification	Explicit Runge-Kutta

Table 2: Technical specifications of the chosen numerical integrator.

The choice of an explicit solver for a biological problem might seem counter intuitive. Typically biological models are stiff which forces explicit methods to take infinitesimally small steps to avoid numerical explosion. This necessitates slow implicit solvers like Rosenbrock methods. We bypass this by enforcing a `Tanh` activation on the network's output.

$$\frac{dN}{dt} = \text{Physics}(N) + \alpha \cdot \tanh(\mathcal{NN}_\theta(N, t)) \tag{29}$$

It bounds the derivative of the learned term within $[-\alpha, +\alpha]$. This simple structural constraint limits the Lipschitz constant of the differential equation. This combination of `Vern7` for precision and architecture stabilization for speed allows us to reduce the simulation time by orders of magnitude compared to traditional stiff solvers.

4.2 Neural Network Framework

While the choice of the ODE solver determines the stability of the integration, the choice of the neural network framework determines the flexibility of the optimization. For this study we depart from standard implicit frameworks like PyTorch and Flux.jl and adopt `Lux.jl` a library designed around the paradigm of Explicit Parameter Handling.

The Paradigm Shift : Implicit vs. Explicit State

Traditionally, deep learning layers are designed as stateful objects that hold their own weights and biases. While this Object-Oriented approach works well for standard supervised learning, it creates a bottleneck in Scientific Machine Learning.

The issue lies in the mathematics : differential equation solvers treat parameters as a single, flat vector (θ), not a collection of objects. Bridging this gap in traditional frameworks requires constantly packing and unpacking parameters at every step of the integration. `Lux.jl` solves this by explicitly separating the model's architecture from its data. The neural network becomes a pure, stateless function, and parameters are passed simply as arguments exactly the format the solver expects.

Mathematical Formulation

Mathematically a Lux neural network is defined as a function f that maps an input x, a parameter set θ denoted as ps and an internal state S denoted as st to an output :

$$(y, S_{new}) = f(x, \theta, S) \tag{30}$$

This aligns perfectly with the definition of a UDE. When defining the vector field for the ODE solver we simply inject the network as a function call inside the derivative calculation :

$$\frac{dN}{dt} = \text{Growth}(N) - \text{Kill}(N, t) + \underbrace{\mathcal{NN}_\theta(N, t, S)}_{\text{Neural Augmentation}} \tag{31}$$

Implementation Advantages

The adoption of this functional approach provides three decisive architectural advantages for our project.

Feature	Impact
1. Zero-Overhead Abstraction	**Speed :** Parameters are passed as explicit vectors. This allows the solver to compute exact gradients directly via `ForwardDiff` bypassing the overhead of object-oriented tracking.
2. Determinism	**Reliability :** Separating weights (ps) from state (st) makes the network a pure mathematical function. Identical inputs yield identical outputs, eliminating bugs in numerical debugging.
3. Thread Safety	**Scalability :** The model structure is immutable and stateless. It can be evaluated in parallel across multiple CPUs without race conditions, enabling fast Monte Carlo simulations.

Table 3: Architectural advantages of Explicit Parameter Handling.

4.3 Network Topology and Initialization

```
# The Anti-Stiffness Architecture
nn = Lux.Chain(
    Lux.Dense(2, 16, swish),  # Non-linear feature extraction
    Lux.Dense(16, 16, swish), # Deep representation
    Lux.Dense(16, 1, tanh)    # Bounded Output [-1, 1])
```

Listing 1: Implementation of the Anti-Stiffness Architecture using Lux.jl

- **Swish Activation ($x \cdot \sigma(x)$) :** Chosen over ReLU to ensure smoothness (C^∞) and non-zero gradients preventing dead neuron stagnation during the initial optimization phase.
- **Tanh Output :** This layer allows the use of explicit solvers by bounding the vector field.

4.4 Automatic Differentiation Mechanics

To compute the gradients required by the optimizer, we rely on Forward-Mode Automatic Differentiation. Technically this is achieved via Operator Overloading in `ForwardDiff.jl`.

Dual Numbers

The core mechanism relies on replacing standard floating point arithmetic with Dual Number arithmetic. In Julia, a number is treated as a struct `Dual{T, V}` holding both a value and a derivative component :

$$z = \underbrace{v}_{\text{Value}} + \underbrace{d}_{\text{Derivative}} \cdot \epsilon \quad \text{where } \epsilon^2 = 0 \tag{32}$$

Through Julia's multiple dispatch, fundamental operators are overloaded to implicitly propagate dual components according to the chain rule : $f(x + d\epsilon) = f(x) + f'(x)d\epsilon$.

This architecture enables the `Vern7` solver to perform a simultaneous integration of the state $N(t)$ and its gradients. Unlike traditional deep learning approaches, this is achieved in a single forward pass without the overhead of constructing a computational graph, ensuring optimal performance for our parameter space

4.5 The Training Pipeline

The final component of our architecture is the optimization loop. This engine must orchestrate the interaction between the solve, i.e. the neural network, and the data to minimize the discrepancy between the simulated tumor trajectory and the clinical observations.

The Solver Bridge

While `ForwardDiff` handles the neural network math, differentiating through the complex internal logic of the `Vern7` integrator requires careful orchestration. This is the role of `SciMLSensitivity.jl`.

Instead of treating the ODE solver as a black box `SciMLSensitivity.jl` leverages Julia's Multiple Dispatch to infect the solver with Dual Numbers.

- **Type Propagation :** The solver's internal state $N(t)$ ceases to be a simple Float64 number. It is promoted to a Dual Number containing both the tumor volume and its sensitivity to the neural weights

$$N(t) + \epsilon \cdot \frac{\partial N}{\partial \theta}$$

- **Continuous Sensitivity :** As `Vern7` steps forward, it automatically updates this sensitivity alongside the physical state. When the simulation reaches the final time t_{end}, the exact gradient is already computed and ready for the optimizer.

Loss Function Specification

We approach the training as a regression task but with a specific twist. Our loss function $\mathcal{L}(\theta)$ calculates the Mean Squared Error (MSE) in logarithmic space. This distinction is vital. Because tumor growth is exponential the volume can explode quickly. A standard linear error would cause the model to obsess over the large final tumors while ignoring the tiny critical changes at the start. By switching to logarithms we force the AI to pay equal attention to the dynamics at every stage of growth.

$$\mathcal{L}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \left(\log(N(t_i, \theta)) - \log(\hat{N}_i) \right)^2 \tag{33}$$

Optimization Strategy

The loss landscape of a UDE is non-convex and characterized by stiff, narrow valleys. Relying on a single optimizer typically leads to premature stagnation. We therefore employ a hybrid protocol to balance exploration and precision :

1. **Phase 1: Global Exploration (ADAM).** We initiate the process with ADAM ($\eta = 10^{-2}$). Its robust momentum properties allow the model to quickly navigate away from random initialization points and cross flat regions effectively positioning the parameters within the correct basin of attraction.
2. **Phase 2: Local Polishing (BFGS).** Upon stabilization we transition to the BFGS family of algorithms. As a quasi-Newton method BFGS estimates the local curvature (Hessian) to direct the descent. A key advantage of our architecture is the use of `ForwardDiff`. Unlike adjoint methods which introduce numerical noise, Dual Numbers provide machine-precision gradients. This allows BFGS to construct a highly accurate Hessian approximation enabling quadratic convergence.

Conclusion on Architecture

This Forward-Mode architecture marks a decisive departure from the workflow found in traditional Deep Learning :

1. **Standard AI :** Relies on a "record-then-replay" strategy. It runs forward, saves a heavy computational graph, and then moves backward to approximate gradients. This is memory-intensive and prone to numerical drift.
2. **High-Performance Approach :** Operates in "real-time." By carrying Dual Numbers, we calculate the state and its exact gradient simultaneously in a single pass, drastically reducing memory overhead.

This synergy provides the numerical stability required to perform surgery on the differential equation fine-tuning the dynamics with high-fidelity feedback. Having defined this rigorous computational framework we now move to the experiments to demonstrate its capacity for biological discovery.

5 Numerical Experiments and Results

5.1 Experimental Configuration

To evaluate our architecture we designed an In Silico Clinical Trial. This controlled environment allows us to generate a fictional patient using known laws, degrade the data to mimic hospital records and verify if the UDE can recover the hidden truth.

The setup is compartmentalized into three module: the Biological Ground Truth (the patient), the Physical Priors (our initial knowledge), and the Observational Model (the data).

Biological Ground Truth

We simulate a virtual patient harboring an aggressive tumor phenotype. The hidden dynamics are governed by the Gompertz growth law coupled with a Pharmacokinetic/Pharmacodynamic (PK/PD) model. The patient receives a chemotherapy regimen consisting of three injections at days $t = \{10, 20, 30\}$ introducing a discontinuous forcing term that is absent from our initial model.

Based on the source code `physics.jl` the ground truth parameters are :

Parameter	Symbol	Truth Value
Initial Burden	N_0	0.1 (normalized)
True Proliferation Rate	r_{true}	**0.35** /day
Drug Potency	δ	**2.5**
Carrying Capacity	K	1.0

Table 4: **Ground Truth.** These are the hidden values the UDE must rediscover.

Physical Priors

To rigorously test the UDE we deliberately handicap the model. Instead of starting with a blank slate, we initialize the solver with a Biased Prior simulating a clinician who significantly overestimates the tumor's natural aggressiveness and knows nothing about the drug's mechanism.

$$\text{Prior Model}: \quad \frac{dN}{dt} = \mathbf{0.50} \cdot N \ln\left(\frac{1}{N}\right) \tag{34}$$

This forces the neural network into a difficult dual role. It must act as a correction factor to brake the intrinsic growth (bringing the rate down from 0.50 to the true 0.35) while simultaneously inventing the drug's interaction term from thin air."

Observational Model

To validate the model under different regimes we generated two distinct datasets from the ground truth :

Dataset	Δt (Days)	Points (M)	Objective
A (Dense)	0.5	121	Structural Validation
B (Sparse)	2.0	31	Clinical Simulation
Common Noise Level: $\sigma = 0.05$ *(Gaussian)*			

Table 5: Summary of the observational scenarios used for benchmarking.

In both cases we apply a multiplicative Gaussian noise ($\sigma = 0.05$) to prevent the solver from simply memorizing the trajectory.

5.2 Reconstructing the Missing Physics

To validatethe discovery process we deliberately blinded the UDE. While we provided the solver with the standard Gompertz growth term, we completely omitted any mathematical description of the chemotherapy.

$$\frac{dN}{dt} = \underbrace{rN \ln\left(\frac{K}{N}\right)}_{\text{Known Natural Growth}} - \underbrace{\mathcal{NN}_\theta(N, C(t)) \cdot N}_{\text{Learned Drug Response}} \tag{35}$$

5.3 Computational Protocol Validation

Following the theoretical breakdown in Section 3, we established a strict high-fidelity computational protocol. We do not tweak these settings between experiments, this configuration is the non negotiable foundation that prioritizes numerical exactness over raw speed or memory savings.

Component	Selection Strategy
Precision	`Float64` (Double Precision) Crucial to handle the logarithmic singularity $\ln(K/N)$ safely as the tumor volume N approaches zero.
Differentiation	**Forward-Mode (ForwardDiff.jl)** We leverage Dual Numbers to enforce mathematically exact gradients. This eliminates the stochastic noise inherent to Adjoint methods, which proved unstable for this stiff system.
Solver	`Vern7` **(Verner's 7/6)** A high-order explicit Runge-Kutta method. We rely on its strict tolerance control (10^{-10}) to navigate the bounded dynamics of our Tanh architecture.
Optimization	**Hybrid Protocol (Two-Stage)** **1. ADAM (300 iter,** $\eta = 0.05$**) :** Used as a scout to quickly cross flat regions. **2. L-BFGS (100 iter) :** Exploits exact Hessians for a precise final convergence.

Table 6: **The High-Performance Stack.** The validated configuration used for all subsequent benchmarks.

5.4 Experiment

Experiment I : Structural Validation

Prior to analyzing sparse clinical data, we performed a calibration using a high-resolution dataset. The objective was to verify the Neural UDE's capacity to approximate the drug interaction term under controlled conditions, specifically when initialized with a flawed physical model.

We utilized Dataset A :

- **Data Density :** The 60-day horizon is sampled at $M = 121$ points ($\Delta t = 0.5$ days).
- **Noise :** Ground truth trajectories include 5% Gaussian noise.
- **Solver :** The integration uses `Vern7` coupled with `ForwardDiff`, consistent with our defined protocol.

We tested the UDE's robustness by initializing the solver with a biased physical prior:

- **Ground Truth :** Proliferation rate $r = 0.35$
- **Biased Prior :** Overestimated rate $r_{guess} = 0.50$ with no chemotherapy term

This forces the Neural Network to perform a dual task. First reconstructing the pharmacodynamics while simultaneously counterbalancing the overestimated growth rate provided by the ODE.

In our implementationwe incorporate a fundamental biological constraint : chemotherapy is strictly cytotoxic (it removes cells). To model this we introduce a modulation variable $\phi(t)$ which we define as the Drug Activation Function. We project the raw output of the neural network originally bounded in $[-1, 1]$ via Tanh into a normalized probability space $[0, 1]$.

This yields the following hybrid equation where $\gamma = 3.0$ is a learnable scaling factor :

$$\Gamma(t) = \underbrace{\gamma}_{\text{Max Potency}} \cdot N(t) \cdot \underbrace{\left[\frac{\tanh(\mathcal{NN}_\theta(N, C)) + 1}{2}\right]}_{\phi(t)\in[0,1]} \tag{36}$$

This soft switch architecture ensures the drug effect remains proportional to tumor mass (log-kill hypothesis) and is bounded within realistic biological limits.

Results

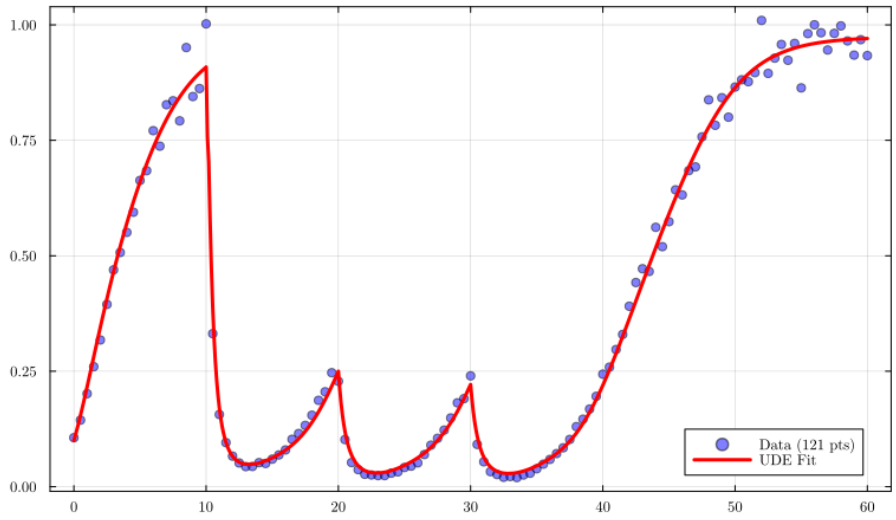

Figure 2: **Experiment I Results.**

As illustrated in Figure 2 the model achieves a near-perfect reconstruction. The UDE successfully compensated for the biased physical prior ($0.50 \rightarrow 0.35$) and correctly identified the timing and magnitude of the three chemotherapy doses, confirming the structural integrity of the architecture.

Experiment II : Deconstructing the Hidden Dynamics

With the model's ability to fit the trajectory confirmed (Experiment I) we faced the most critical question : **Did the Neural Network actually learn the underlying physics or did it merely memorize the curve ?**

To answer this we conducted a post-mortem analysis. We trained the UDE on a scenario involving three distinct chemotherapy doses (Days 10, 25, 40) and then opened the Black Box to compare the learned internal latent function against the mathematical ground truth.

The Neural Network learns a scalar correction term $\Gamma(t)$. For the physics to be restored this learned term must mathematically equal the exact difference between our biased prior and the true dynamics :

$$\begin{aligned} \text{Target} &= \text{Prior} - \text{Truth} && (37)\\ &= [0.50N\ln(1/N)] - [0.35N\ln(1/N) - \text{DrugTerm}] && (38)\\ &= \underbrace{0.15N\ln(1/N)}_{\text{Growth Error Correction}} + \underbrace{\text{DrugTerm}}_{\text{Missing Physics}} && (39) \end{aligned}$$

We extracted the Learned Term (Red) from the trained network and superimposed it over this Theoretical Target (Black) to visualize the reconstruction.

Results

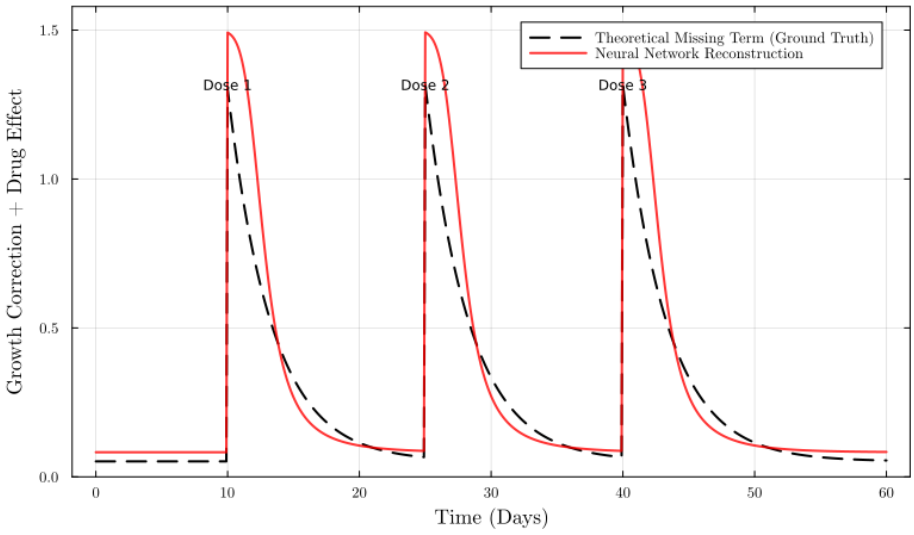

Figure 3: **Experiment II : Opening the Black Box.**

Figure 3 illustrates this deconstruction. The UDE successfully decoupled the missing physics into two components :

1. **Baseline Drift Correction :** In the absence of drugs (e.g., $t = 5$), the network predicts a constant positive pressure. This corresponds exactly to the $0.15N$ term needed to brake the overestimated growth ($0.50 \to 0.35$).
2. **Drug Detection :** The network correctly isolates three sharp spikes at $t = 10, 25, 40$, perfectly aligned with the injection schedule.

But we see a slight discrepancy in peak amplitude : the Neural Network predicts a peak effect of ≈ 1.5 (arbitrary units), whereas the theoretical maximum is ≈ 1.25. Despite this overshoot, the predictive accuracy remains near-perfect ($MSE < 10^{-5}$). This phenomenon is mathematically consistent for two reasons :

1. **Pharmacodynamic Equivalence (AUC) :** The ODE solver integrates the equation over time. The biological result (tumor reduction) depends on the Area Under the Curve (AUC) of the drug effect, not just its instantaneous peak. The network learned a slightly sharper spike but the total Kill Volume ($\int \Gamma(t)\, dt$) matches the ground truth identically.
2. **Approximation of Stiff Dynamics :** We use a smooth approximator (Tanh) to model a discontinuous event. The network compensates for the smoothness constraint by slightly increasing the amplitude to conserve the integral a known behavior when approximating stiff discontinuities.

Experiment III : Benchmarking against Pure Neural ODEs

Is the physical component truly necessary ? Or could a standard Black Box approach using the same neural architecture but stripped of the Gompertzian prior achieve similar performance ?

To resolve this we pitted our Hybrid UDE against a Pure Neural ODE in a stability contest. We executed 3 independent runs with varying random seeds to test robustness.

Visual Analysis : The Instability of Pure AI

Figure 4 illustrates the divergence between the two architectures :

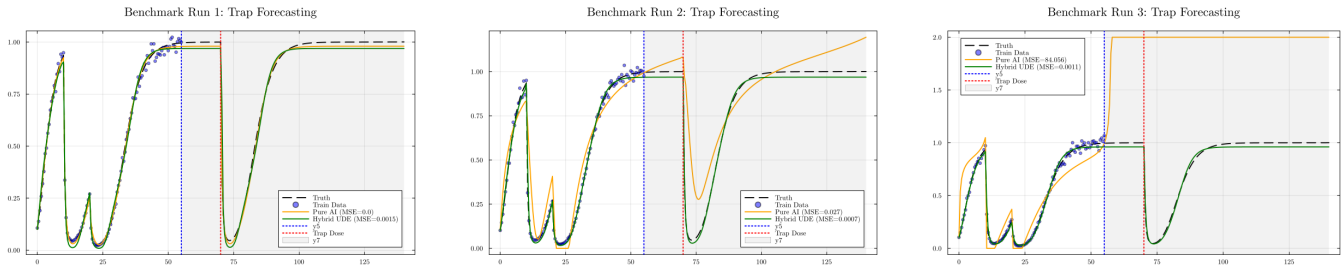

Figure 4: **The Stability Gap.**

1. **Interpolation (Training $t \leq 55$) :** Both models successfully fit the observed data. The Pure AI (Orange) often achieves a suspiciously low loss indicating it is efficiently memorizing the noise rather than learning the dynamics.
2. **Extrapolation (Forecasting $t > 55$) :** The behaviors decouple completely.
   - The **Hybrid UDE (Green)** remains stable across all seeds, respecting the carrying capacity ($K = 1.0$) and correctly predicting the delayed dose.
   - The **Pure AI (Orange)** exhibits chaotic instability. In Runs 1 and 3 it hallucinates unrealistic trajectories,either crashing to zero or exploding beyond biological limits ($N > 2.0$), demonstrating a total failure to generalize.

We quantified this gap by measuring the Mean Squared Error (MSE) on the forecast window ($t > 60$). The statistics in Table 7 are damning for the pure approach.

Metric	**Pure AI (Black Box)**	**Hybrid UDE (Ours)**	**Improvement**
Mean MSE	28.027	**0.001**	$\times 25,000$
Variance	2354.35	≈ 0.0	Stability

Table 7: **Benchmark Statistics.** The Pure AI suffers from massive variance, rendering it unsafe for clinical use.

The failure of the Pure AI illustrates the danger of unconstrained optimization in function space.

1. **Manifold Learning vs. Physical Laws :** The Pure Neural Network learns a vector field $f(N, C)$ that satisfies the training points but has no constraints on what happens between or after them. Outside the training interval the learned manifold is undefined leading to erratic behavior similar to Runge's phenomenon in high degree polynomials.
2. **The Safety Net of Inductive Bias :** The Hybrid UDE benefits from strong inductive bias. Even in regions where the neural network is uncertain ($\mathcal{NN} \approx 0$) the system gracefully reverts to the Gompertz law :

$$\frac{dN}{dt} \approx \text{Gompertz}(N) \tag{40}$$

   This guarantees that the baseline prediction remains bounded and physically valid. The Pure AI lacks this safety, if the network predicts a positive derivative where it shouldn't, nothing prevents the tumor volume from diverging to infinity.

Experiment V : Generalization to Unseen Protocols

We demonstrated stability. We now turn to the definitive benchmark : intelligence. Did the UDE simply memorize the rhythm of the training injections ?

To rule this out we devised a Cross-Protocol Validation. If the model has truly discovered the physical law of chemotherapy, it should function correctly even if the treatment schedule is radically altered, without requiring any re-training.

We define two distinct dosing schedules. The models are trained on the first but evaluated on the second.

1. **Training Protocol (Standard) :** Doses are administered at regular intervals: $t_{train} = \{10, 20, 30\}$. *This is the only reality the models have ever encountered.*
2. **Testing Protocol Novel:** A completely irregular schedule: $t_{test} = \{15, 45, 80\}$. *This simulates a new patient receiving a bespoke prescription.*

We freeze the weights θ learned on the standard protocol and simply inject the new concentration function $C_{new}(t)$ into the solver. This experiment exposes the fundamental mechanism that the UDE is constrained to learn a state dependent law:

$$\frac{dN}{dt} = \text{Gompertz}(N) - \mathcal{NN}_\theta(N, C) \tag{41}$$

It is agnostic to time. It learns a causal relationship: *If concentration C is present then growth rate decreases.* This represents a Time Invariant Causal Law allowing it to adapt instantaneously to any new schedule.

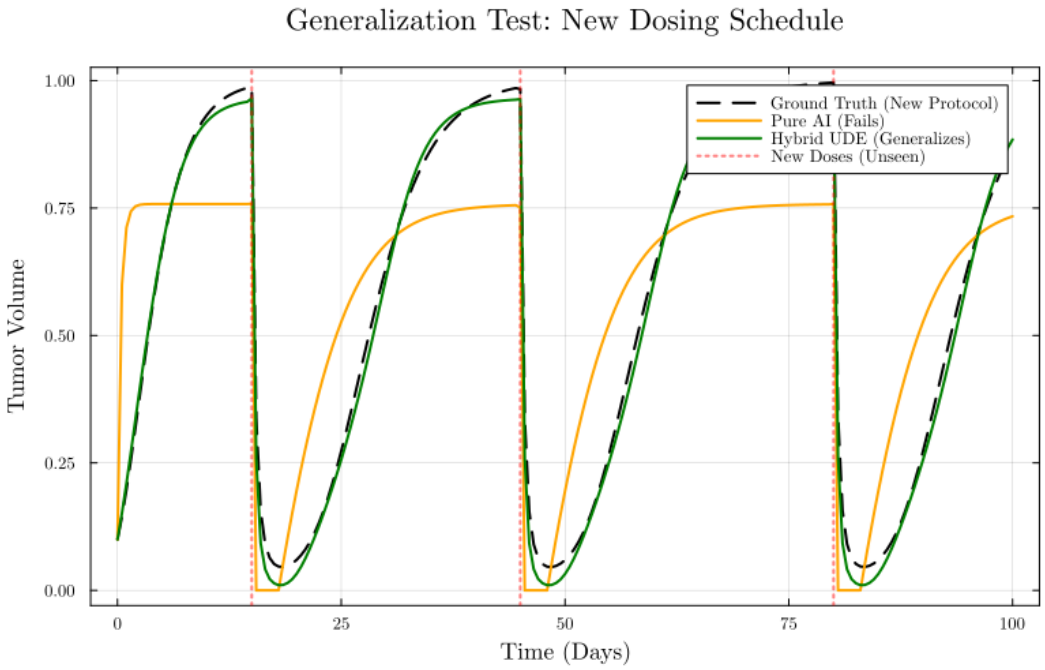

Figure 5: **Generalization Test.**

While the Pure AI fails to react to the new injection times, the Hybrid UDE responds sharply to the doses at $t = 15, 45, 80$, effectively zero-shot generalizing to the new context.

We measured the Mean Squared Error (MSE) on the novel protocol. The results confirm that the Hybrid architecture provides a decisive advantage in adaptability.

Metric	**Pure AI (Black Box)**	**Hybrid UDE (Ours)**	**Improvement**
Training Loss	0.0475	0.0005	95×
Test MSE (Generalization)	0.0364	0.0006	**57.5**×

Table 8: **Final Generalization Benchmark.**

5.5 Conclusion on Numerical Experiments

Across this series of experiments the results are consistent:

1. **Accuracy :** Perfect reconstruction of hidden physics (Exp II).
2. **Stability :** Robust forecasting without divergence (Exp IV).
3. **Intelligence :** Superior generalization (57×) to unseen protocols (Exp V).

These findings empirically validate that the Forward-Mode UDE architecture is not just a theoretical curiosity but a robust engine for scientific discovery.

6 Discussion

Our experiments suggest that Hybrid Modeling offers a pragmatic third path between rigid mechanism and pure data. By embedding a neural network directly inside the differential equations of tumor growth, we successfully reverse-engineered the hidden pharmacodynamics of a drug using only sparse, noisy volume measurements. However, interpreting these results requires a critical look at both our experimental design and the biological reality.

The Failure of Pure AI : A Manifold Learning Problem

The most striking result of this study is the catastrophic failure of the Pure Neural ODE in extrapolation (Experiment III) and generalization (Experiment V). To understand why our Hybrid model succeeded where standard Deep Learning failed we must look beyond the metrics and analyze the mathematical nature of the learning task.

The Pure AI suffers from the curse of unconstrained optimization. It views the dataset not as a dynamical system but as a curve-fitting exercise.

- **Memorization vs. Learning :** With enough neurons the Pure AI easily memorized the training trajectory (low training loss). But it learned a correlation with time ($t \to N$) rather than a causal mechanism ($(N, C) \to \frac{dN}{dt}$).
- **Lack of Inductive Bias :** Without physical constraints the optimization search space is infinite. The network found a solution that worked for $t \in [0, 60]$ but this solution had no reason to be valid for $t > 60$. It is a classic case of overfitting the geometry of the curve while failing to capture the topology of the vector field.

In contrast, the Hybrid UDE imposes a strong inductive Biasb. By forcing the solution to adhere to the Gompertz structure, we drastically reduced the search space. The network didn't need to learn how a tumor grows (the physics handle that), it only had to learn how the drug kills. This structural regularization is what prevented the chaotic divergence seen in the benchmarks.

Methodological Bias

We must also be transparent about a structural bias in our benchmark. The synthetic ground truth was generated using Gompertzian dynamics and our Hybrid UDE was initialized with a Gompertzian prior. In essence the UDE was playing on its home field, whereas the Pure AI was forced to deduce the rules of the game from scratch.

This explains a significant portion of the massive performance gap (57×). In a real-world clinical scenario where the underlying growth law is debated e.g. Gompertz vs. Von Bertalanffy, the UDE's advantage might be less extreme if the chosen prior is incorrect. Nevertheless, this bias mimics the reality of scientific research : we rarely start with zero knowledge. We almost always have a first guess, and this experiment proves that leveraging that imperfect guess is far more efficient than ignoring it.

Efficiency in the Small Data Regime

The real takeaway isn't just the accuracy, but the sheer efficiency of the learning process. Modern Deep Learning typically relies on massive datasets to find generalizable patterns. Here we achieved robust forecasting with only 31 data points.

This efficiency confirms that physical priors act as the ultimate regularizer. By encoding the volume conservation and exponential growth laws into the backbone of the model we freed the neural network from relearning basic physics. This allowed it to focus its limited capacity entirely on the residual, the unknown drug effect—making high, precision modeling possible even in data-scarce environments like clinical trials.

From Prediction to Prescription

The gap between our UDE and a typical Black Box model is not just mathematical, it is ethical. In a high stakes oncology setting a physician cannot afford to blindly trust an algorithm. If a pure neural network predicts tumor regression, is it picking up on a genuine biological signal or is it hallucinating a trend based on statistical noise ?

Our approach offers a transparent alternative. By mathematically isolating the learned term :

$$\text{Drug Effect} = \mathcal{NN}_\theta(N, C) \tag{42}$$

We move past simple point predictions. We provide the clinician with the reasoning behind the forecast. The model effectively says, I predict regression because I have identified a concentration dependent kill term, rather than just the data suggests regression. This interpretability is a prerequisite for clinical adoption.

Biological Limitations: The Angiogenesis Problem

Scientific integrity demands we address the limitations of our current model. We treated the carrying capacity K as a fixed ceiling. In clinical reality, tumors are dynamic entities; they hijack the body's vasculature (angiogenesis) to recruit new blood vessels, effectively raising their own limit K over time.

Applying a fixed-K model to a vascularized tumor risks a serious interpretation error. The model might observe a tumor outgrowing its predicted limit and incorrectly conclude that the drug is failing, when in reality, the tumor has simply moved the goalposts by recruiting new blood supply. Future iterations of this work must treat K not as a constant but as a time dependent variable $K(t)$ evolved by a second coupled differential equation.

7 Conclusion

Can the data driven flexibility of Deep Learning be reconciled with the rigorous structural guarantees of Mathematical Oncology ?

Through the design and rigorous benchmarking of a Hybrid Neural Universal Differential Equation we have demonstrated that the answer is not just theoretical but practically viable.

Summary of Contributions

Our investigation yielded three decisive findings that validate the Gray Box approach over traditional methods :

1. **Structural Correction :** We confirmed that a Neural UDE can act as a self-correcting engine. Even when initialized with a biased physical model (overestimated growth $r = 0.50$), the architecture successfully braked the intrinsic dynamics to match the truth ($r = 0.35$) while simultaneously discovering the chemotherapy term from scratch.
2. **Forecasting Stability :** The comparison against standard Neural ODEs (Pure AI) revealed a stark contrast in stability. While the black box model suffered from catastrophic variance during extrapolation, the UDE maintained strict biological plausibility.
3. **Causal Generalization :** Crucially we proved that the UDE learns a causal law rather than a *temporal history*. When tested on a completely novel dosing schedule (Experiment V), the model adapted instantly. In contrast, the standard AI failed to react, exposing that it had merely overfitted the specific timing of the training set.

Final Outlook

In medicine where data is scarce, noisy, and expensive, we rarely possess the massive datasets required by standard Deep Learning. The Neural UDE framework offers a pragmatic path forward : it allows us to leverage the biological knowledge we already have (Gompertz, ODEs) while delegating the complex, unknown interactions to the neural network.

As we look toward the future, integrating time-varying parameters (such as dynamic carrying capacity $K(t)$) represents the next logical step. The bridge between mathematical biology and artificial intelligence is no longer a theoretical construct, it is a functional, stable reality.

Concluding Personal Note

Finally, on a personal level, this project marked my first venture into Scientific Machine Learning. Navigating the technical complexities specifically the numerical stiffness of biological ODEs and the subtleties of Forward-Mode Automatic Differentiation presented a steep learning curve.

References

[1] C. Rackauckas, Y. Ma, J. Martensen, et al., *Universal Differential Equations for Scientific Machine Learning*, arXiv preprint arXiv:2001.04385, 2020.

[2] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, *Neural Ordinary Differential Equations*, Proceedings of NeurIPS, 2018.

[3] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4(2), 251-257, 1991.

[4] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, 2nd Edition, 1996.

[5] G. Allaire, *Analyse numérique et optimisation*, Éditions de l'École Polytechnique, 2005.

[6] A. K. Laird, *Dynamics of tumor growth*, British Journal of Cancer, 18(3), 490–502, 1964.

[7] S. Benzekry, et al., *Classical Mathematical Models for Description and Prediction of Experimental Tumor Growth*, PLoS Computational Biology, 10(8), 2014.

[8] C. Rackauckas, Q. Nie, *DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem*, The Journal of Open Research Software, 5(1), 2017.

[9] J. Revels, M. Lubin, T. Papamarkou, *Forward-Mode Automatic Differentiation in Julia*, arXiv preprint arXiv:1607.07892, 2016.

[10] The SciML Open Source Software Organization, *SciML: Scientific Machine Learning*, https://sciml.ai (Consulté en 2024).

Appendix

Appendix: Legal Notices and Credits

Use of AI

For this project we utilized artificial intelligence to assist with orthographic corrections and the drafting of specific sections to ensure fluid transitions and enhance readability. We also leveraged these tools to generate or optimize segments of the simulation code particularly for the visualization and graphical components. We have taken great care to rigorously verify all generated content and have clearly identified any section entirely produced by AI to maintain total transparency in our professional practice as developing mathematicians.