Supplementary Text: data simulation and details of the primary neural network

1 Outliers of the subject data

Throughout this study, we consider subjects 3,11,18 and 25 as outliers and exclude their physiological data from the procedures described in Section 2. These subjects are excluded due to the distinct characteristics of their FFA data, particularly in terms of monotonicity and range. Including these outliers would complicate the generation of simulated data, the training of the neural network, and the accurate inference of parameters from the trained network. Figure S1 (see in supplementary figures; applies to all Figures S2 to S14) illustrates the FFA physiological data for all 25 subjects. The black curves correspond to subjects 11, 18, and 25, whose FFA values are notably low near the end of the time course. Meanwhile, the brown curve represents subject 3, whose FFA values are disproportionately high over the interval $[50, 80]$.

2 Parameter optimization for models

For the 3D model, the optimization is carried out by minimizing the dimensionless cost function defined as:

$$C_{\mathrm{3D}} =_{t\in t_{\mathrm{post20}}}^{\mathrm{mean}} \left[\frac{(G(t) - G_{\mathrm{data}}(t))^2}{G_{\mathrm{data}}(t)^2} + \frac{(F(t) - F_{\mathrm{data}}(t))^2}{F_{\mathrm{data}}(t)^2} \right], \tag{1}$$

where $G(t)$ and $F(t)$ represent the glucose and FFA trajectories, respectively, obtained from the integration of the ODE model using parameter values from each optimization step. $G_{\mathrm{data}}(t)$ and $F_{\mathrm{data}}(t)$ denote the physiological glucose and FFA data. At every iteration, given the C_X value and interpolation of $I_{\mathrm{data}}(t)$, we integrate the equation

$$\frac{\mathrm{d}X}{\mathrm{d}t} = C_X(\max\{I(t) - i_b, 0\} - X(t)), X(0) = 0, \tag{2}$$

up to $t = 22$ to compute X_{22} for the subsequent iteration.

Similarly, for the 2D model, we minimize the cost function

$$C_{\mathrm{2D}} =_{t\in t_{\mathrm{all}}}^{\mathrm{mean}} \frac{(F(t) - F_{\mathrm{data}}(t))^2}{F_{\mathrm{data}}(t)^2}. \tag{3}$$

Figures S2, S3 and S4 show the fittings of glucose and FFA for all 21 subjects with the 3D and 2D models, respectively.

All optimizations were performed on a local workstation equipped with a 2.6 GHz 6-Core Intel® Core™ i7 processor and 16 GB of 2400 MHz DDR4 RAM within a Jupyter Notebook environment using SciPy's Nelder–Mead algorithm. Taking the 3D model as example, across all 25 subjects, the optimization converged in an average of 857 iterations (range: 576 to 2000, where 2000 is the maximal number of iteration), requiring approximately 1335 objective-function evaluations per subject. The mean wall-clock time per subject was 300.2 s (total runtime for all 25 subjects: 125.1 min).

3 Data simulation with Gaussian Process Regression

The Gaussian Process Regression(GPR) model we use has Radial Quasi-Periodic (RQ) kernel:

$$k_{\mathrm{RQ}}(\mathbf{x}_1, \mathbf{x}_2) = \left(1 + \frac{1}{2\alpha}(\mathbf{x}_1 - \mathbf{x}_2)^{\mathrm{T}}\Theta^{-2}(\mathbf{x}_1 - \mathbf{x}_2)\right)^{-\alpha}, \tag{4}$$

where Θ is a lengthscale parameter with a prior in $(0.01, 5)$, and the rational quadratic relative weighting parameter has prior $(0.8, 1)$. With this kernel, we define the exact marginal log likelihood as

$$\mathrm{mll}(\mathbf{x}_1, \mathbf{x}_2) = -\frac{1}{2}(\mathbf{x}_1 - \mu)^{\mathrm{T}}K^{-1}(\mathbf{x}_2 - \mu) - \frac{1}{2}\log\det(K) - \frac{n}{2}\log(2\pi), \tag{5}$$

where $K_{ij} = k_{\mathrm{RQ}}(x_1^{(i)}, x_2^{(j)})$. Suppose we have physiological data of insulin from N subjects, normalize them using the mean and standard deviation at each individual time point, and denote the normalized data as $\{I_{\mathrm{Nmlz}}^{(i)}(t) \mid t \in t_{\mathrm{vec}}\}_{i=1}^{N}$, where $t_{\mathrm{vec}} = t_{\mathrm{all}}$ or t_{post20}. During the training of this GPR model M, each time we randomly choose a subject index i, and calculate the value and backpropagate gradients of the following loss function L

$$L = -\mathrm{mll}(M(t_{\mathrm{vec}}), I_{\mathrm{Nmlz}}^{(i)}(t_{\mathrm{vec}})). \tag{6}$$

To maximize the marginal log likelihood, we iterate this process and update the model parameters to minimize L until it is lower than a predefined threshold (we set it as 0.73 for the FSIGT physiological data used in this study). To achieve this, we employ the Adam optimizer with a learning rate of 0.001.

4 Sampling of parameter set

Here we put the details of generating sample parameter set. For each parameter p, its optimized values are obtained from the data of all subjects. Additionally, before integrating the model, we must generate samples for G_b, F_b, $G(22)$(initial condition of G), and $F(22)$(initial condition of F). These values are directly obtained from the physiological data of each subject. In the remaining sections of this paper, we treat G_b and F_b as parameters of 3D model, and F_b as a parameter of the 2D model. Notably, I_b is the first value of the insulin data generated using Gaussian process regression (GPR), so I_b does not require sampling at this stage.

We use a multivariate log-normal distribution to fit the parameter sets from optimization, and generate as many parameter samples as required. Figure S5 and S6 show the correlation between the optimized parameters and generated samples for these two models respectively.

After generating the simulated data, we filter out parameter-trajectory pairs where the glucose or FFA trajectories contain negative values. For the 3D model, we also remove cases where the trajectory is not monotonically decreasing over the interval [22,30], which includes five time points. This filtering step ensures that the dataset remains consistent with expected physiological patterns.

5 Training of the neural network

We generate a simulated dataset using the method described in the main text. The dataset is randomly split, with 80% allocated for training and 20% for validation. To ensure consistency and reproducibility, a fixed random seed is used during the splitting process. The training data is shuffled to prevent order bias, while the validation set remains fixed for consistent evaluation. Mini-batches are employed to enhance computational efficiency and training stability.

A separate testing dataset is generated using the same procedure as the training and validation datasets. This testing dataset is never used during training or validation, ensuring an unbiased assessment of the trained model. The same testing dataset is applied across all models for fair comparison.

Before training, each column of the neural network output d_O is linearly rescaled to the range $[0, 1]$. During the inference step after the training is complete, the rescaling is inversely applied to the outputs to recover the original parameter values.

The neural network is trained using the Adam optimizer with the learning rate Lr schedule given by the following function:

$$\mathrm{Lr}(\mathrm{epoch}) = \begin{cases} \mathrm{MaxLr} \times \mathrm{C}_3 / \left(1 + \exp\left(-\mathrm{C}_1 \cdot (\mathrm{epoch} - \mathrm{N}_{\mathrm{top}}/2)\right)\right) & , \mathrm{epoch} \leq \mathrm{N}_{\mathrm{top}}, \\ \mathrm{MaxLr} \times \mathrm{C}_4 / \left(1 + \exp\left(-\mathrm{C}_2 \cdot \left(\frac{3}{4}\mathrm{N}_{\mathrm{max}} - \mathrm{epoch}\right)\right)\right) & , \mathrm{epoch} > \mathrm{N}_{\mathrm{top}}, \end{cases} \tag{7}$$

where the maximal learning rate MaxLr, epoch number N_{top} where learning rate arrives at maximum, and number of epoch N_{max} for the training are to be determined, and $C_1 = 10/N_{\mathrm{top}}$, $C_2 = 10/N_{\mathrm{max}}$. $C_3 = 1 + \exp\left(-C_1 N_{\mathrm{top}}/2\right)$ and $C_4 = 1 + \exp\left(-C_2\left(\frac{3}{4}N_{\mathrm{max}} - N_{\mathrm{top}}\right)\right)$ are such that this function is continuous at N_{top}. The training process span 1000 epoches, and the model with the lowest validation loss is saved. For the training of the network, we set maximum learning rate of 10^{-3}, batch size 500, $N_{\mathrm{max}} = 2000$ and $N_{\mathrm{top}} = 1000$.

The implementation of our neural network framework is conducted using Python 3.9 and PyTorch 2.1.2. All model training, parameter inference, and evaluations are performed within this computational environment, ensuring consistency and reproducibility.

6 Feature Engineering and neural network architecture

Let N be the batch size. We first denote some layers as follows which we will use in feature engineering:

1. MaxPooling2D, a pooling layer with kernel size=$(1, 2)$;
2. Flatten, input size=$(N, 1, N_1, N_{\mathrm{conv}})$, output size=$(N, N_1 N_{\mathrm{conv}})$;
3. Dense_1: dense layer with input size same as the output size of Flatten layer, and output size=(N, N_2), employing the hyperbolic tangent function (tanh) as its activation function;
4. Dense_2: dense layer with input size (N, N_2) and output size=(N, N_3), employing the hyperbolic tangent function (tanh) as its activation function;
5. Dense_3: the final dense layer to output d_O, with input size=(N, N_3) and output size=(N, N_{para}). It uses ReLU or $\frac{1}{2}(\tanh(x) + 1)$ as final activation function, considering that all parameters are positive before and after rescaling.

Here we show the details of several features:

1. No Feature Engineering:

   d_I is processed by a two-dimensional convolutional layer (conv2D) called $\mathrm{conv2D}_1$ with kernel size=(2,2) and stride=(2,1), followed by a Maxpooling2D layer. Then it is processed by another 2D convolutional layer called $\mathrm{conv2D}_2$ with kernel size=(6,2), followed by a Maxpooling2D layer. The data is then flattened and fed into two dense layers Dense_1 and Dense_2. The input and output sizes of these layers are put in the Table 1.

2D model	$\mathrm{conv2D}_1$	$\mathrm{conv2D}_2$	Dense_1	Dense_2
input size	$(N, 12, 28, 1)$	$(N, 6, 13, 1)$	$(N, 3072)$	$(N, 1024)$
output size	$(N, 6, 27, 1)$	$(N, 1, 12, 1)$	$(N, 1024)$	$(N, 1024)$
3D model	$\mathrm{conv2D}_1$	$\mathrm{conv2D}_2$	Dense_1	Dense_2
input size	$(N, 12, 16, 1)$	$(N, 6, 7, 512)$	$(N, 1536)$	$(N, 1024)$
output size	$(N, 6, 15, 512)$	$(N, 1, 6, 512)$	$(N, 1024)$	$(N, 1024)$

Table 1. Input and output sizes for layers for the 2D and the 3D models for feature 1: no feature engineering. The four numbers for conv2D layers represent, respectively, the batch size, the numbers of rows, the numbers of columns, and the layer dimension.

2. Concatenation: d_I has the same structure as in Feature 1 (No Feature Engineering), with an additional concatenation process: the input data d_I is initially processed by a two-dimensional convolutional layer conv2D$_1$, with kernel size=(2,2), stride=(2,1), using padding of replication on the right. Subsequently, its output is concatenated with d_I to incorporate the functions of time and substances (glucose, FFA, and insulin), as well as their derivatives with respect to time. This concatenation is performed by pairing the 12 rows of d_I into 6 pairs, and after each pair, inserting the corresponding row from the output of conv2D$_1$. The concatenated data then passes through two conv2D layers (denoted as conv2D$_2$ and conv2D$_3$, respectively) each followed by a Maxpooling2D layer. conv2D$_2$ has kernel size=(3,2), stride=(3,1); conv2D$_3$ has kernel size=(6,2). The data is then flattened and fed into two dense layers as before. The full process is illustrated in Figure S9, and the input and output sizes of layers are put in Table 2.

2D model	conv2D$_1$	conv2D$_2$	conv2D$_3$	Dense$_1$	Dense$_2$
input size	$(N, 12, 28, 1)$	$(N, 18, 28, 512)$	$(N, 6, 27, 512)$	$(N, 3072)$	$(N, 1024)$
output size	$(N, 6, 28, 512)$	$(N, 6, 27, 512)$	$(N, 1, 13, 512)$	$(N, 1024)$	$(N, 1024)$
3D model	conv2D$_1$	conv2D$_2$	conv2D$_3$	Dense$_1$	Dense$_2$
input size	$(N, 12, 16, 1)$	$(N, 18, 16, 512)$	$(N, 6, 15, 512)$	$(N, 1536)$	$(N, 1024)$
output size	$(N, 6, 16, 512)$	$(N, 6, 15, 512)$	$(N, 1, 7, 512)$	$(N, 1024)$	$(N, 1024)$

Table 2. Input and output sizes for layers for the 2D and the 3D models for feature 2: concatenation. The four numbers for conv2D layers represent, respectively, the batch size, the numbers of rows, the numbers of columns, and the layer dimension.

7 Denoising from physiological data

The neural network we train takes trajectory data of ODE systems as input. However, real physiological data are not smooth like the model-generated solutions. Therefore, it is necessary to denoise the physiological data before using it as input for the neural network. To achieve this, we employ a simple fully connected neural network for denoising, and prepare the training data as follows:

- The output of the network is the simulated trajectory set obtained in the earlier section.
- The input for each output trajectory is generated by adding noise to the simulated trajectory data.

Here we show how we extract adjacent noise information from the physiological data, by taking the FFA data over N time points as an example. For subject k, let

$$\Delta F(k,l) = \begin{cases} \left| \frac{F_{\text{data}}(k,l+1) - F_{\text{data}}(k,l)}{F_{\text{data}}(k,l+1) + F_{\text{data}}(k,l)} \right|, & \text{if } l = 1, \\ \frac{1}{2} \left[\left| \frac{F_{\text{data}}(k,l+1) - F_{\text{data}}(k,l)}{F_{\text{data}}(k,l+1) + F_{\text{data}}(k,l)} \right| + \left| \frac{F_{\text{data}}(k,l) - F_{\text{data}}(k,l-1)}{F_{\text{data}}(k,l) + F_{\text{data}}(k,l-1)} \right| \right], & \text{if } 2 \le l \le N-1, \\ \left| \frac{F_{\text{data}}(k,l) - F_{\text{data}}(k,l-1)}{F_{\text{data}}(k,l) + F_{\text{data}}(k,l-1)} \right|, & \text{if } l = N \end{cases} \tag{8}$$

denote the adjacent increment at the l-th point over the time course. We add 20 percent noise to each of the output of the neural network, using the following formula:

$$F_{\text{noise}}(k,l) = \begin{cases} F_{\text{data}}(k,l), & \text{if } l = 1, \\ F_{\text{data}}(k,l) + 0.2Y \left[F_{\text{data}}(k,l-1) + F_{\text{data}}(k,l) \right], & \text{if } l \ge 2, \end{cases} \tag{9}$$

where $Y \sim \mathcal{N}(0, \overset{\text{mean}}{k} (\Delta F(k,l)))$ is sampled from a normal distribution with a mean of 0 and a standard deviation equal to the mean of adjacent increment $\Delta F(k,l)$ across all subjects k. This noisy trajectory serves as the corresponding input data for training the neural network.

We use the FFA data from the first 200,000 training samples as the output and add noise to these samples using the above method to generate the corresponding input data. A fully connected neural network is then trained with this data. The network consists of three dense layers, each with 256 nodes followed by the ReLU activation function. The loss function is set to be the mean squared error (MSE) between the inferred outputs and the given outputs. The network is trained using the learning rate schedule defined by (7), with $N_{\max} = 100$ and $N_{\text{top}} = 50$.

Once the training is complete, we feed the glucose and FFA physiological data from all subjects into the trained network, which outputs the corresponding denoised data. Figure S14 illustrates the denoised FFA data for all subjects.