```julia
# --- Cell 1: Setup ---

using DifferentialEquations
using Flux
using Plots
using Optimization
using OptimizationOptimisers
using Zygote
using DataFrames

# Hodgkin-Huxley Model Parameters (Global Constants)
const Cm = 1.0        # μF/cm^2
const g_Na = 120.0    # mS/cm^2
const g_K = 36.0      # mS/cm^2
const g_L = 0.3       # mS/cm^2
const E_Na = 50.0     # mV
const E_K = -77.0     # mV
const E_L = -54.387   # mV
```

```
-54.387
```

```julia
# --- Cell 2: Known Physics & Stimulus ---

# Voltage-gated ion channel kinetics
α_n(V) = 0.01 * (V + 55) / (1 - exp(-(V + 55) / 10))
β_n(V) = 0.125 * exp(-(V + 65) / 80)
α_m(V) = 0.1 * (V + 40) / (1 - exp(-(V + 40) / 10))
β_m(V) = 4.0 * exp(-(V + 65) / 18)
α_h(V) = 0.07 * exp(-(V + 65) / 20)
β_h(V) = 1 / (1 + exp(-(V + 35) / 10))

# Steady-state & time-constant functions for the 2D model
m_inf(V) = α_m(V) / (α_m(V) + β_m(V))
h_inf(V) = α_h(V) / (α_h(V) + β_h(V))
n_inf(V) = α_n(V) / (α_n(V) + β_n(V))
tau_n(V) = 1 / (α_n(V) + β_n(V))

# Stimulus protocol: a short, sharp pulse
function stimulus(t; start=10.0, duration=1.0, amplitude=20.0)
    (start <= t < start + duration) ? amplitude : 0.0
end
```

```
stimulus (generic function with 1 method)
```

```julia
# --- Cell 3: Data Generation ---

# 2D Hodgkin-Huxley reduced model engine
function hodgkin_huxley_reduced!(du, u, p, t)
    V, n = u
    I_ext = stimulus(t)
    I_Na = g_Na * m_inf(V)^3 * h_inf(V) * (V - E_Na)
```

```
    I_K  = g_K  * n^4 * (V - E_K)
    I_L  = g_L * (V - E_L)
    du[1] = (I_ext - I_Na - I_K - I_L) / Cm
    du[2] = (n_inf(V) - n) / tau_n(V)
end

# Define and solve the 2D problem to generate training data
u₀_reduced = [-65.0, 0.3177]
tspan_reduced = (0.0, 50.0)
prob_reduced = ODEProblem(hodgkin_huxley_reduced!, u₀_reduced,
tspan_reduced, nothing)
sol_reduced = solve(prob_reduced, Tsit5(), saveat=0.1)

# Extract and structure the training data
const training_data_2D = Array(sol_reduced)
const timestamps_2D = sol_reduced.t

# (Optional) Verify data shape and content
df = DataFrame(t=timestamps_2D, V=training_data_2D[1, :],
n=training_data_2D[2, :])
println("Generated Training Data:")
display(first(df, 5))
```

```
WARNING: redefinition of constant Main.training_data_2D. This may
fail, cause incorrect answers, or produce other errors.
WARNING: redefinition of constant Main.timestamps_2D. This may fail,
cause incorrect answers, or produce other errors.
```

```
5×3 DataFrame
 Row │ t        V         n
     │ Float64  Float64   Float64
─────┼──────────────────────────
   1 │     0.0  -65.0     0.3177
   2 │     0.1  -64.9997  0.3177
   3 │     0.2  -64.9994  0.317699
   4 │     0.3  -64.9991  0.317699
   5 │     0.4  -64.9989  0.317699
```

```
Generated Training Data:
```

```
# --- Cell 4: UDE Definition ---

# Define the Neural Network structure
const U = Chain(Dense(1, 10, tanh), Dense(10, 1))

# Extract the trainable parameters (p_nn) and the re-structuring
function (re)
p_nn, re = Flux.destructure(U)

# Define the UDE function with the embedded neural network
function ude_model!(du, u, p_nn, t)
```

```
    V, n = u

    # Neural network component to learn the unknown current
    unknown_current = re(p_nn)([V])[1]

    # Known physics components
    I_ext = stimulus(t)
    I_K  = g_K  * n^4 * (V - E_K)
    I_L  = g_L * (V - E_L)

    # The hybrid dynamics equation
    du[1] = (I_ext + unknown_current - I_K - I_L) / Cm
    du[2] = (n_inf(V) - n) / tau_n(V)
end
```

```
WARNING: redefinition of constant Main.U. This may fail, cause
incorrect answers, or produce other errors.
```

```
ude_model! (generic function with 1 method)
```

```
using SciMLSensitivity

# --- Cell 5: Loss Function ---


function predict_ude(p_nn)
    prob_ude = ODEProblem(ude_model!, u₀_reduced, tspan_reduced, p_nn)
    prediction = solve(prob_ude, Tsit5(), saveat=timestamps_2D,
                       sensealg=InterpolatingAdjoint(autojacvec=true))
    return prediction
end

# Loss function: quantifies the error between the UDE prediction and
the ground truth data
function loss_function(p_nn)
    prediction = predict_ude(p_nn)

    # Penalty for unsuccessful simulations
    if prediction.retcode != :Success
        return Inf
    end

    # Sum of squared errors
    loss = sum(abs2, Array(prediction) .- training_data_2D)
    return loss
end
```

```
loss_function (generic function with 1 method)
```

```
# --- Cell 6: Training ---
```

```julia
# Callback function to display progress during training
losses = Float64[]
iter = 0
callback = function (p, l)
    global iter += 1
    push!(losses, l)
    if iter % 10 == 0
        println("Iteration $iter | Current Loss: $l")
    end
    return false # Must return false to continue training
end

# Define the optimization problem
optf = Optimization.OptimizationFunction((x, p) -> loss_function(x),
Optimization.AutoZygote())
optprob = Optimization.OptimizationProblem(optf, p_nn)

# Execute the training mission
println("Commencing Training...")
# We use a lower learning rate for stability and more iterations.
# This is a full-scale training run. It may take a few minutes.
result_ude = Optimization.solve(optprob, Adam(0.01), callback =
callback, maxiters = 1000)

println("--- TRAINING COMPLETE ---")
println("Final Loss: $(result_ude.objective)")
```

```
Commencing Training...

┌ Warning: Layer with Float32 parameters got Float64 input.
│   The input will be converted, but any earlier layers may be very
slow.
│   layer = Dense(1 => 10, tanh)
│   summary(x) = 1-element Vector{Float64}
└ @ Flux C:\Users\Admin\.julia\packages\Flux\uRn8o\src\layers\
stateless.jl:60

Iteration 10 | Current Loss: 1909.5606353800993
Iteration 20 | Current Loss: 1626.3829034125367
Iteration 30 | Current Loss: 1843.152045797983
Iteration 40 | Current Loss: 2.742347230065062
Iteration 50 | Current Loss: 0.31103337843553847
Iteration 60 | Current Loss: 3265.392364924261
Iteration 70 | Current Loss: 0.07137205529573457
Iteration 80 | Current Loss: 1603.2203007398632
Iteration 90 | Current Loss: 1458.1770466191565
Iteration 100 | Current Loss: 0.6726953129927876
Iteration 110 | Current Loss: 1989.4604466036244
Iteration 120 | Current Loss: 0.3267610638802199
Iteration 130 | Current Loss: 1443.6341678015776
```

```
Iteration 140 | Current Loss: 2181.5336595603367
Iteration 150 | Current Loss: 0.2718774556298728
Iteration 160 | Current Loss: 1494.2836054531163
Iteration 170 | Current Loss: 0.16234353888819192
Iteration 180 | Current Loss: 0.622883386882508
Iteration 190 | Current Loss: 2156.7360179785874
Iteration 200 | Current Loss: 3.294376672750254
Iteration 210 | Current Loss: 0.13793089043072523
Iteration 220 | Current Loss: 0.06460104060586085
Iteration 230 | Current Loss: 2017.224763301653
Iteration 240 | Current Loss: 2.8433701924237633
Iteration 250 | Current Loss: 2314.592203446586
Iteration 260 | Current Loss: 1967.4892451923204
Iteration 270 | Current Loss: 2142.6890197551543
Iteration 280 | Current Loss: 1527.787107972351
Iteration 290 | Current Loss: 1552.0321418048072
Iteration 300 | Current Loss: 1297.8197196312328
Iteration 310 | Current Loss: 3.2088164912981774
Iteration 320 | Current Loss: 2744.4264705350047
Iteration 330 | Current Loss: 0.07219013814404748
Iteration 340 | Current Loss: 2150.034306620229
Iteration 350 | Current Loss: 2.833163337954611
Iteration 360 | Current Loss: 1970.4446143909156
Iteration 370 | Current Loss: 2094.350046879548
Iteration 380 | Current Loss: 1882.2757736289566
Iteration 390 | Current Loss: 1801.1778204674615
Iteration 400 | Current Loss: 2.7118382550636135
Iteration 410 | Current Loss: 1402.8549033135373
Iteration 420 | Current Loss: 3.727176630400065
Iteration 430 | Current Loss: 2.229579058187025
Iteration 440 | Current Loss: 1.7331524011853363
Iteration 450 | Current Loss: 1851.7599990314743
Iteration 460 | Current Loss: 1.31743512012212
Iteration 470 | Current Loss: 1725.894191312443
Iteration 480 | Current Loss: 8.049736140184997
Iteration 490 | Current Loss: 31.418706027315366
Iteration 500 | Current Loss: 1550.2716334741528
Iteration 510 | Current Loss: 4.93987961636695
Iteration 520 | Current Loss: 6.249638143412842
Iteration 530 | Current Loss: 1087.3702740061624
Iteration 540 | Current Loss: 1507.6247336400545
Iteration 550 | Current Loss: 1816.5738451531504
Iteration 560 | Current Loss: 1482.5415367491387
Iteration 570 | Current Loss: 2226.862862801957
Iteration 580 | Current Loss: 1649.31103504041
Iteration 590 | Current Loss: 1214.9340743633024
Iteration 600 | Current Loss: 2117.998481380647
Iteration 610 | Current Loss: 0.7445333404807156
Iteration 620 | Current Loss: 1506.193631575973
```

```
Iteration 630 | Current Loss: 1393.386577583562
Iteration 640 | Current Loss: 2341.792442600628
Iteration 650 | Current Loss: 1554.9125261427514
Iteration 660 | Current Loss: 1271.420817894922
Iteration 670 | Current Loss: 1447.5108214845684
Iteration 680 | Current Loss: 1118.861065873432
Iteration 690 | Current Loss: 1.1665059260611117
Iteration 700 | Current Loss: 922.2288297952546
Iteration 710 | Current Loss: 1108.6751318496497
Iteration 720 | Current Loss: 1205.342389205871
Iteration 730 | Current Loss: 1389.0641961037338
Iteration 740 | Current Loss: 1349.9104663681258
Iteration 750 | Current Loss: 2.055518609156949
Iteration 760 | Current Loss: 974.4329321987715
Iteration 770 | Current Loss: 1420.3593596943904
Iteration 780 | Current Loss: 11.226912486660604
Iteration 790 | Current Loss: 1924.6337344585575
Iteration 800 | Current Loss: 1008.2885871347619
Iteration 810 | Current Loss: 1277.6837226225264
Iteration 820 | Current Loss: 1019.7902262271953
Iteration 830 | Current Loss: 943.4310970741726
Iteration 840 | Current Loss: 1635.1351543729784
Iteration 850 | Current Loss: 1328.467912581975
Iteration 860 | Current Loss: 1504.2761738214158
Iteration 870 | Current Loss: 0.02074506544151123
Iteration 880 | Current Loss: 1221.7050119278806
Iteration 890 | Current Loss: 1358.1947807932263
Iteration 900 | Current Loss: 794.051481163178
Iteration 910 | Current Loss: 1199.6404396142452
Iteration 920 | Current Loss: 1110.9988092479152
Iteration 930 | Current Loss: 11.658632487929198
Iteration 940 | Current Loss: 1200.4683341551545
Iteration 950 | Current Loss: 1240.9940090962148
Iteration 960 | Current Loss: 1626.711292444194
Iteration 970 | Current Loss: 50.33992254474068
Iteration 980 | Current Loss: 928.174973988135
Iteration 990 | Current Loss: 6.518240620619786
Iteration 1000 | Current Loss: 11.662406981619165
--- TRAINING COMPLETE ---
Final Loss: 0.005343778045629235
```

```julia
plot(losses,
     xlabel="Iteration",
     ylabel="Loss",
     title="Training Loss Over Iterations",
     label="Loss",
     lw=2,
     yscale=:log10)
```


Training Loss Over Iterations
Loss
10^3
10^0
Loss
0
250
500
750
1000
Iteration