

Title: BAND: A Unified Probabilistic Framework for Non-Stationary Heart Rate Variability Analysis.

Authors: Matías Castillo-Aguilar^{1,2}, David Medina-Ortiz^{3,4}, Ruby Méndez Muñoz^{1,2}, Diego Mabe-Castro^{1,2}, Noah Beelders⁵, Atenea Uribe-Ojeda^{1,2}, Marcelo A. Navarrete^{1,2} and Cristian Núñez-Espinosa^{1,2*}.

¹ Centro Asistencial Docente y de Investigación (CADI-UMAG), Punta Arenas, Chile.

² Escuela de Medicina, Universidad de Magallanes (UMAG), Punta Arenas, Chile.

³ Departamento de Ingeniería en Computación, Universidad de Magallanes, Punta Arenas, Chile.

⁴ Centre for Biotechnology and Bioengineering, CeBiB, Universidad de Chile, Santiago, Chile.

⁵ Department of Mathematics, University of Munich, Theresienstraße 39, 80333 Munich, Germany.

****Correspondence***

Cristian Núñez-Espinosa, Escuela de Medicina, Universidad de Magallanes, Punta Arenas, Chile. Centro Asistencial de Docencia e Investigación CADI-UMAG, Chile. e-mail: cristian.nunez@umag.cl. Address: Avenida Bulnes 01855, Box 113-D. Phone: +56 61 2201411.

This document provides supplementary information, extended analyses, and technical details to support the findings presented in the main manuscript. The goal of this supplement is to ensure full methodological transparency, computational reproducibility, and to provide a comprehensive validation of the proposed model.

Section S1: Model Implementation & Priors

This section details the full implementation of the model, including the complete model code, justification for all prior distributions, and an analysis of the model's sensitivity to these choices.

Section S1.1: Fully Commented Stan Model Code

Below is the complete, line-by-line commented code for the generative model implemented in the Stan programming language. The comments explain the function of each parameter, transformation, and sampling statement.

```
// Stan Implementation of the Full Generative R-R Interval Model
//
// This model deconstructs an R-R interval time series into several components:
// 1. A time-varying baseline (mean R-R interval).
// 2. A time-varying total standard deviation (SDNN).
// 3. A structured, oscillatory signal representing physiological variability.
// 4. An unstructured residual noise component.
// The model uses a double-logistic form to capture a perturbation-recovery dynamic.

// =====
// Functions Block
// =====
// This block defines custom functions that can be used elsewhere in the Stan program.
// Defining functions here is efficient, especially for repeated calculations.
functions {
  /**
   * Calculates a standard logistic growth curve (sigmoid).
   * This function is a fundamental building block for modeling dynamic changes over time.
   * @param t The time vector.
   * @param location The time point of the curve's inflection point (midpoint).
   * @param rate The steepness or rate of the transition.
   * @return A vector of values between 0 and 1 representing the logistic curve.
   */
  vector logistic_curve(vector t, real location, real rate) {
    return inv_logit(rate * (t - location));
  }
}

// =====
// Data Block
// =====
// This block declares all the data that will be passed into the model from
// an external source (e.g., R or Python).
data {
  // --- Observed Data ---
  int<lower=1> N; // Number of data points in the time series.
  vector[N] t; // Time vector (e.g., in minutes).
```

```

vector[N] RR; // Observed R-R intervals (e.g., in ms).

// --- Fixed Spectral Components (treated as data) ---
// The model uses a fixed basis of sine and cosine waves.
// The frequencies are provided, and the model estimates their amplitudes.
int<lower=1> N_sin; // Number of sinusoids per frequency band.
array[3] vector[N_sin] freqs; // Pre-calculated frequencies for VLF, LF, and HF bands.

// --- Informative Priors on Double-Logistic (DL) parameters ---
// Passing these as data allows for setting subject-specific or experiment-specific
// priors, which can greatly improve model identifiability.
real<lower=0> tau_mu; // Expected time of the perturbation event.
real<lower=0> delta_mu; // Expected delay between perturbation and recovery.
real<lower=0> lambda_mu; // Expected rate of the perturbation.
real<lower=0> phi_mu; // Expected rate of the recovery.
}

// =====
// Transformed Data Block
// =====
// This block pre-computes quantities that depend only on the data.
// Performing these calculations here is highly efficient, as they are done
// only once before sampling begins, rather than on every iteration.
transformed data {
  // === Data-derived scaling parameters ===
  // These are used to scale parameters to a more natural range (e.g., a standard
  // normal scale), which makes defining priors and sampling more robust.
  real rr_min = min(RR);
  real rr_range = max(RR) - rr_min;
  real rr_sd = sd(RR);
  real t_min = min(t);
  real t_range = max(t) - t_min;

  // --- Precompute sine and cosine basis function templates ---
  // These matrices hold the values of  $\sin(2\pi f t)$  and  $\cos(2\pi f t)$  for all
  // time points and all basis frequencies. This is a major performance optimization.
  array[3] matrix[N, N_sin] sin_mat;
  array[3] matrix[N, N_sin] cos_mat;

  // Precomputed reciprocal of N, used to center each basis column (subtract
  // its mean) below.
  real inv_N = 1.0 / N;

  for (j in 1:3) {
    // Convert time from minutes to seconds for frequency calculation (Hz).
    matrix[N, N_sin] T_mat = (t * 60) * freqs[j];
    matrix[N, N_sin] sin_raw = sin(2 * pi() * T_mat);
    matrix[N, N_sin] cos_raw = cos(2 * pi() * T_mat);

    // Center each basis column (sinusoid) BEFORE forming the Gram matrices
    // below. This guarantees (a) the synthesized oscillator signal
    //  $S_j(t) = \sin\_mat * u\_sin + \cos\_mat * u\_cos$  is automatically mean-zero for
    // ANY coefficients, and (b) the Gram matrices are computed from the
    // exact same centered basis that generates the signal, so the variance
    // computed from them ( $\Sigma_S$ , below) is actually exact. Previously the
    // basis was centered only after the fact (see Section 5), so  $\Sigma_S$ 
    // described the variance of the UNCENTERED signal while  $S\_t\_matrix$  held
    // the centered one -- a mismatch.
    row_vector[N_sin] sin_col_means = (rep_row_vector(1.0, N) * sin_raw) * inv_N;
    row_vector[N_sin] cos_col_means = (rep_row_vector(1.0, N) * cos_raw) * inv_N;
    sin_mat[j] = sin_raw - rep_matrix(sin_col_means, N);
    cos_mat[j] = cos_raw - rep_matrix(cos_col_means, N);
  }

  // --- Precompute log-frequencies ---
  // The GP prior operates on the log of the frequencies to better handle
  // the wide range of frequencies often seen in HRV.
  array[3] vector[N_sin] log_freqs;
  array[3] vector[N_sin] log_freqs_scaled;
  array[3] real log_range;
  for (j in 1:3) {
    log_freqs[j] = log(freqs[j]);
    log_range[j] = max(log_freqs[j]) - min(log_freqs[j]);
  }
}

```

```

    // Work on scaled frequency range to make rho parameter scale-agnostic
    log_freqs_scaled[j] = (log_freqs[j] - min(log_freqs[j])) / log_range[j];
}

// --- Precompute per-band Gram matrices ---
// These matrices contain the inner products of the basis vectors (e.g., sin_i * sin_j).
// They are used later to calculate the *exact* sample variance of the synthesized
// oscillatory signals, without making the simplifying (and often incorrect)
// assumption that the basis functions are perfectly orthogonal over the discrete
// time interval.
array[3] matrix[N_sin, N_sin] G_sin;
array[3] matrix[N_sin, N_sin] G_cos;
array[3] matrix[N_sin, N_sin] G_sin_cos; // For cross-terms between sin and cos.

// Normalization factor for sample variance (1 / (N-1)).
real normalization = 1.0 / (N - 1);

for (j in 1:3) {
    G_sin[j] = (transpose(sin_mat[j]) * sin_mat[j]) * normalization;
    G_cos[j] = (transpose(cos_mat[j]) * cos_mat[j]) * normalization;
    G_sin_cos[j] = (transpose(sin_mat[j]) * cos_mat[j]) * normalization;
}
}

// =====
// Parameters Block
// =====
// This block declares all the parameters the model will estimate.
// For optimal performance with Stan's HMC sampler, all parameters are declared on
// an unconstrained scale (the real line, -inf to +inf). They are transformed to their
// meaningful, constrained scales in the `transformed parameters` block.
parameters {
    // --- Shared timing/rate parameters for the double-logistic curves ---
    real tau_logit; // Location of the perturbation event (Logit scale).
    real delta_logit; // Delay between perturbation and recovery (Logit scale).
    real lambda_log; // Rate of the perturbation (Log scale).
    real phi_log; // Rate of the recovery (Log scale).

    // --- Baseline RR(t) trend parameters ---
    real alpha_r_logit; // Initial baseline RR value (Logit-scaled).
    real beta_r_logit; // Magnitude of the RR drop (Logit-scaled).
    real c_r_logit; // Fractional recovery of the RR drop (Logit-scaled).

    // --- Total Variability SDNN(t) trend parameters ---
    real alpha_s_logit; // Initial baseline SDNN value (Logit-scaled).
    real beta_s_logit; // Magnitude of the SDNN drop (Logit-scaled).
    real c_s_logit; // Fractional recovery of the SDNN drop (Logit-scaled).

    // --- Parameters for time-varying frequency proportions p(t) ---
    // These control the transition between a baseline and a perturbed spectral state.
    vector[2] y_base_log; // Unconstrained (ALR) coords for baseline proportions.
    vector[2] y_pert_log; // Unconstrained (ALR) coords for perturbed proportions.
    real c_c_logit; // Fractional recovery of spectral proportions (Logit-scaled).

    // --- GP Hyperparameters ---
    // These control the properties of the smooth function for the spectral shape.
    array[3] real rho_gp_logit; // Length-scale: Controls the smoothness of the spectrum.

    // --- Non-Centered GP Parameter ---
    // Instead of sampling the GP function values directly, we sample standard normal
    // deviates and reconstruct the GP. This is the Non-Centered Parameterization (NCP).
    array[3] vector[N_sin] z_gp; // Standard normal deviates for the GP.

    // --- Non-Centered Oscillator Coefficients ---
    // The final amplitudes for each sine/cosine pair are also non-centered.
    // We estimate standard normal deviates (`z`), which are then scaled.
    array[3] vector[N_sin] z_sin; // Standard normal deviates for sine coefficients.
    array[3] vector[N_sin] z_cos; // Standard normal deviates for cosine coefficients.

    // --- Fractional split of SDNN variance ---
    real w_logit; // Proportion of total variance that is "structured" (Logit scale).
}

```

```

// =====
// Transformed Parameters Block
// =====
// This block constructs the full generative model step-by-step from the parameters.
// ALL calculations here are performed at every Leapfrog step of the sampler.
transformed parameters {
  // Declare variables that will be used in the likelihood calculation.
  vector[N] mu;           // The final predicted mean RRI trajectory.
  vector[N] var_resid;    // The final residual variance trajectory.

  // --- 0. Map unconstrained parameters to their meaningful scales ---
  // This section applies inverse transformations (e.g., inv_logit, exp) to the
  // parameters, mapping them from the unconstrained real line back to their
  // natural, interpretable scales (e.g., time, ms, proportions).
  real tau = inv_logit(tau_logit) * t_range + t_min;
  real delta = inv_logit(delta_logit) * (t_range - tau); // Delta is a fraction of remaining time.
  real lambda = exp(lambda_log);
  real phi = exp(phi_log);

  real alpha_r = inv_logit(alpha_r_logit) * 2 * rr_range + rr_min;
  real beta_r = (inv_logit(beta_r_logit) * 2 - 1) * alpha_r; // Beta is a fraction of Alpha.
  real c_r = inv_logit(c_r_logit) * 2; // Recovery can be up to 200% (overshoot).

  real alpha_s = inv_logit(alpha_s_logit) * rr_sd;
  real beta_s = (inv_logit(beta_s_logit) * 2 - 1) * alpha_s; // Beta is a fraction of Alpha.
  real c_s = inv_logit(c_s_logit) * 2; // Recovery can be up to 200% (overshoot).

  real c_c = inv_logit(c_c_logit); // Spectral recovery is between 0-100%.
  real w = inv_logit(w_logit); // Structured variance fraction is between 0-1.

  array[3] real rho_gp; // Constrained smoothness of GP
  for (j in 1:3) {
    rho_gp[j] = inv_logit(rho_gp_logit[j]);
  }

  // --- 1. Construct the two logistic building blocks ---
  // These shared curves drive all the dynamic changes in the model.
  vector[N] D1 = logistic_curve(t, tau, lambda); // Primary perturbation curve.
  vector[N] D2 = logistic_curve(t, tau + delta, phi); // Recovery curve.

  // --- 2. Construct the baseline and SDNN trajectories ---
  // These define the time-varying mean and total standard deviation of the RR signal.
  // Note: `SDNN_t` here corresponds to sigma_total(t_i) in the main manuscript; we
  // keep the `SDNN_t` identifier in code for brevity, but the manuscript text uses
  // sigma_total(t_i) to make clear it is a latent, instantaneous scale parameter
  // rather than a conventional windowed SDNN sample statistic.
  vector[N] RR_baseline = fmax(1e-8, alpha_r + beta_r .* D1 - (c_r * beta_r) .* D2);
  vector[N] SDNN_t = fmax(1e-8, alpha_s + beta_s .* D1 - (c_s * beta_s) .* D2);

  // --- 3. Construct the master controller C(t) and spectral proportions p_t ---
  // C(t) governs the transition from a baseline spectral state to a perturbed one.
  vector[N] C_t = D1 .* (1.0 - c_c .* D2);

  // Convert the unconstrained `y_*_log` parameters to 3-element proportion vectors
  // using the softmax function (an inverse Additive Log-Ratio transform).
  vector[3] pi_base = softmax(append_row(y_base_log, 0.0)); // Baseline state.
  vector[3] pi_pert = softmax(append_row(y_pert_log, 0.0)); // Perturbed state.

  // `p_t` smoothly interpolates between the two spectral states based on C(t).
  matrix[N, 3] p_t = (1.0 - C_t) * pi_base + C_t * pi_pert;

  // --- 4. Construct the spectral coefficient vectors (u_sin, u_cos) ---
  // This section combines the GP reconstruction and the final coefficient scaling
  // in a single, efficient loop.
  array[3] vector[N_sin] u_sin;
  array[3] vector[N_sin] u_cos;
  array[3] vector[N_sin] log_v; // This will hold the centered GP values.

  for (j in 1:3) {
    // --- Step A: Reconstruct the Centered GP (Non-Centered Parameterization) ---
    // 1. Construct the covariance matrix K from the hyperparameters.
    matrix[N_sin, N_sin] K =

```

```

gp_exp_quad_cov(to_array_1d(log_freqs_scaled[j]), 1, rho_gp[j])
+ diag_matrix(rep_vector(1e-8, N_sin));

// 2. Compute the Cholesky factor of K.
matrix[N_sin, N_sin] L = cholesky_decompose(K);

// 3. Construct the centered log_v by transforming the standard normal `z_gp`.
log_v[j] = L * z_gp[j];

// 4. Build positive amplitudes from GP
vector[N_sin] a_k = exp(log_v[j]);

// expected diagonal contribution to variance under z ~ N(0,1)
vector[N_sin] diag_sum = diagonal(G_sin[j]) + diagonal(G_cos[j]); // length N_sin

// expected base variance under N(0,1) coefficients (cross-terms vanish)
real base_v_diag = dot_product(a_k .* a_k, diag_sum);

// normalize so expected variance = 1 (avoid divide by zero)
vector[N_sin] full_scale = a_k / sqrt(base_v_diag + 1e-12);

// final NCP coefficients
u_sin[j] = z_sin[j] .* full_scale;
u_cos[j] = z_cos[j] .* full_scale;
}

// --- 5. Synthesize the (already mean-centered) oscillator signals ---
// Because sin_mat/cos_mat are themselves mean-centered (see transformed
// data), the resulting S_j is exactly mean-centered by construction; the
// subtraction below is a negligible residual-mean safety check.
matrix[N, 3] S_t_matrix;
for (j in 1:3) {
  vector[N] S_j = sin_mat[j] * u_sin[j] + cos_mat[j] * u_cos[j];
  S_t_matrix[:, j] = S_j - mean(S_j);
}

// --- 6. Calculate the exact variance of each oscillator using Gram matrices ---
// This quadratic form calculates Var(S_j) = u' * G * u efficiently and exactly.
// Sigma_S is treated as diagonal (independence between bands); see Supplementary
// Section S1.4 for an empirical quantification of this approximation.
matrix[3, 3] Sigma_S = rep_matrix(0.0, 3, 3);
for (j in 1:3) {
  real vj = dot_product(u_sin[j], G_sin[j] * u_sin[j])
    + dot_product(u_cos[j], G_cos[j] * u_cos[j])
    + 2 * dot_product(u_sin[j], G_sin_cos[j] * u_cos[j]);
  Sigma_S[j, j] = vj;
}

// --- 7. Deterministic Inversion to find the amplitude A(t) ---
// This is a powerful trick. Instead of trying to sample A(t), we solve for the
// exact amplitude needed to make the variance of the structured signal match the
// target structural variance defined by `SDNN_t` and `w`. This improves efficiency.
vector[N] var_struct = square(SDNN_t) .* w; // Target structured variance.

// The denominator is the variance of the weighted sum of oscillators if A_t were 1.
matrix[N, 3] M = p_t * Sigma_S;
vector[N] denom_sq = rows_dot_product(M, p_t);
vector[N] denom = sqrt(denom_sq + 1e-12);

// The final amplitude is the ratio of the target SD to the current SD.
vector[N] A_t = sqrt(var_struct) ./ denom;

// --- 8. Combine components to get the final predicted mean `mu` ---
vector[N] sum_weighted_S = rows_dot_product(S_t_matrix, p_t);
mu = RR_baseline + A_t .* sum_weighted_S;

// --- 9. Define the residual variance used in the likelihood ---
// The variance not explained by the structured oscillators becomes the residual variance.
var_resid = square(SDNN_t) .* (1.0 - w);
}

// =====
// Model Block

```

```

// =====
// This block specifies the priors for the parameters and the likelihood function
// that connects the model to the observed data.
model {
  // === Priors ===
  // Priors are placed on the unconstrained parameters. This is essential for
  // ensuring the priors cover the entire space the sampler is exploring.

  // --- Priors for logistic components (timing and rates) ---
  tau_logit ~ normal(logit((tau_mu - t_min) / t_range), 0.2);
  delta_logit ~ normal(logit(delta_mu / (t_range - tau_mu)), 0.2);
  lambda_log ~ normal(log(lambda_mu), 0.2);
  phi_log ~ normal(log(phi_mu), 0.2);

  // --- Priors for RR(t) and SDNN(t) parameters ---
  alpha_r_logit ~ normal(0, 2);
  beta_r_logit ~ normal(0, 2);
  c_r_logit ~ normal(0, 2);
  alpha_s_logit ~ normal(0, 2);
  beta_s_logit ~ normal(0, 2);
  c_s_logit ~ normal(0, 2);

  // --- Priors for spectral proportion p_j(t) parameters ---
  y_base_log ~ normal([0, 0]', 2);
  y_pert_log ~ normal([0, 0]', 2);
  c_c_logit ~ normal(1, 2);

  // --- Priors for the Smoothness of the GP ---
  rho_gp_logit ~ normal(0, 1); // Allows more wiggleness

  // --- Priors for Non-Centered Parameters ---
  // As part of the NCP pattern, the "raw" parameters are given standard normal
  // priors. This is the core of what makes the NCP work.
  for (j in 1:3) {
    z_gp[j] ~ std_normal(); // Prior for the non-centered GP.
    z_sin[j] ~ std_normal(); // Prior for the sine coefficients.
    z_cos[j] ~ std_normal(); // Prior for the cosine coefficients.
  }

  // --- Prior for the fractional variance split ---
  w_logit ~ normal(3, 2); // Weakly favors a higher proportion of structured variance.

  // === Likelihood ===
  // This is the core statement that connects the model's predictions (`mu` and
  // `var_resid`) to the actual observed data `RR`. The model learns by trying
  // to find parameter values that make the observed data most plausible under a
  // Normal distribution with a time-varying mean and variance.
  RR ~ normal(mu, sqrt(var_resid));
}

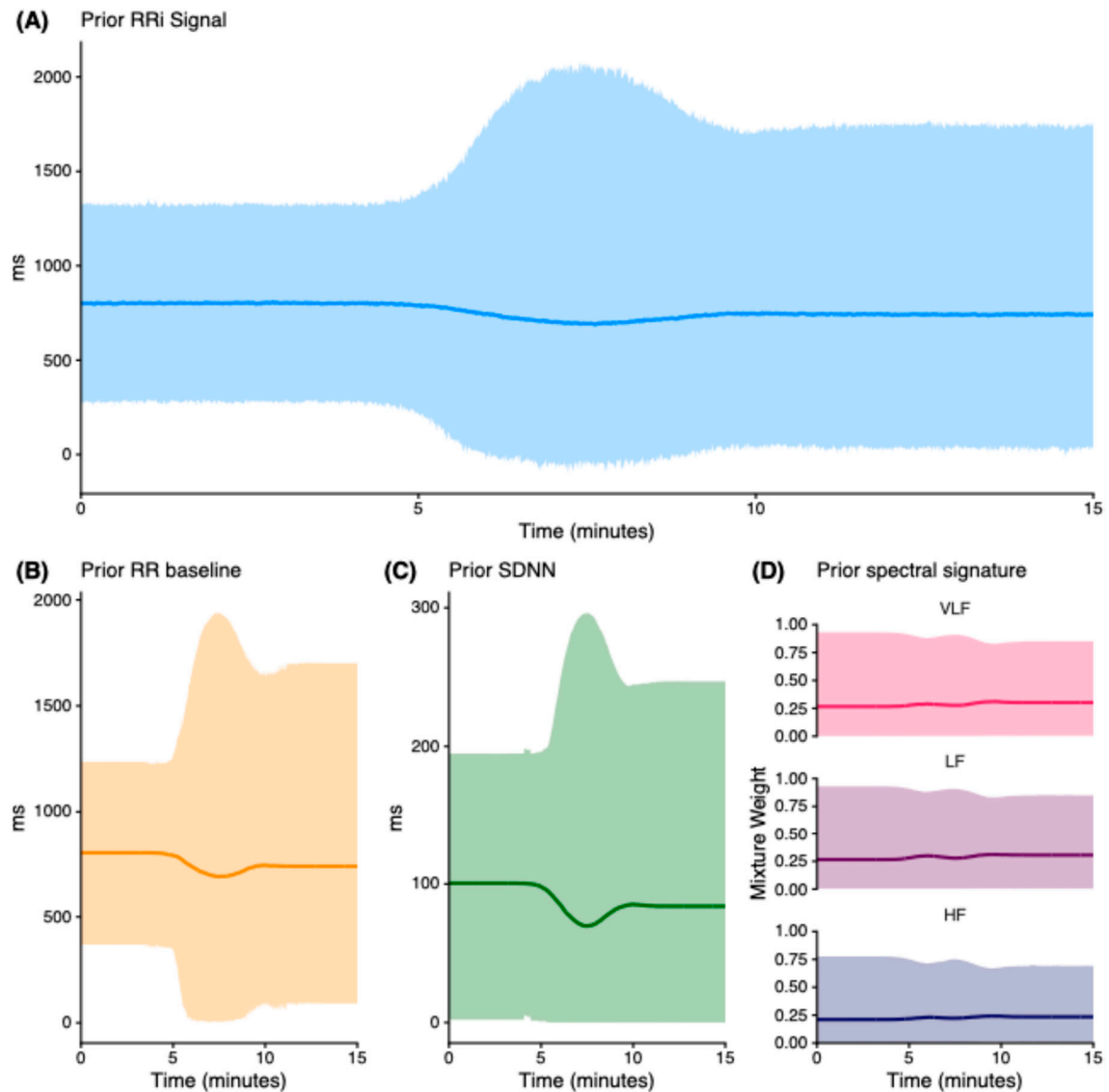
// =====
// Generated Quantities Block
// =====
// This block computes derived quantities from the posterior draws that are not
// needed for sampling itself, but are useful for reporting and downstream
// comparison against external methods (e.g., STFT- or CWT-based power estimates).
generated quantities {
  // p_t (defined in transformed parameters) are mixture/amplitude weights on the
  // oscillators S_j(t_i), not power or variance proportions in themselves. Because
  // Sigma_S[j,j] need not be identical across bands, the actual fractional
  // contribution of band j to the structured variance at each time point, q_j(t_i),
  // is this derived quantity rather than p_t itself; q is the quantity that is
  // properly comparable to STFT- or CWT-derived relative power.
  matrix[N, 3] q_t;
  {
    vector[3] Sigma_S_diag = diagonal(Sigma_S);
    for (i in 1:N) {
      row_vector[3] num = square(p_t[i, :]) .* Sigma_S_diag';
      q_t[i, :] = num / sum(num);
    }
  }
}

```

Section S1.2: Prior Justification and Sensitivity Analysis

Section S1.2.1: Prior Predictive Checks

To ensure our chosen priors were reasonable and did not unduly influence the posterior, we conducted prior predictive checks. We generated 20,000 datasets directly from the prior distributions (i.e., without conditioning on the data). Supplementary Figure S1 shows that the chosen priors generate a wide range of physiologically plausible (and implausible) response curves, confirming that the model is flexible and that the subsequent posterior is primarily driven by the data, not the priors.



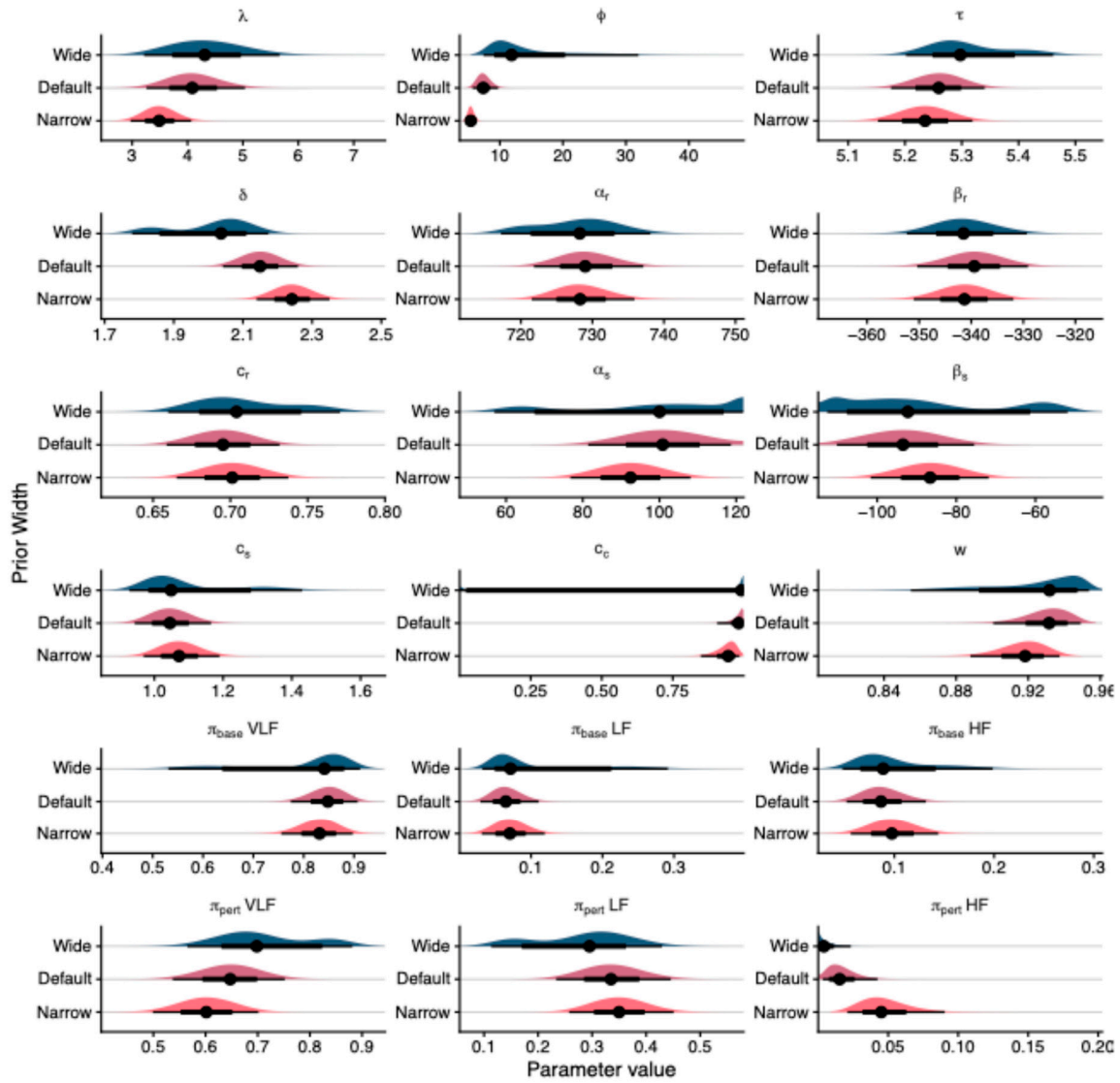
Supplementary Figure S1: Prior Predictive Checks. (A) Simulated RRI signal incorporating deterministic and stochastic components (blue). (B-C) Time-domain reconstruction from prior distributions. (D) Spectral components separated by frequency band. Each line represents the median and the shaded areas represent the 95% highest density interval from 20,000 dataset generated from the prior distribution. The wide distribution of curves demonstrates that the priors are not restrictive.

Section S1.2.2: Prior Sensitivity Analysis

To assess the robustness of our conclusions, we performed a sensitivity analysis by re-fitting the model on the empirical data using three different sets of priors: (1) the main manuscript’s informative priors, (2) “flatter” priors with two times larger variance, and (3) “tighter”, more informative priors with half of original variance. Supplementary Figure S2 shows the posterior distribution for key parameters under each condition. The central conclusion of dissonant recovery (a credible difference between c_r and c_s) remains unchanged across all specifications, with consistent overlapping credible intervals for all model parameters supporting the robustness of our primary finding.

We revise our interpretation of the remaining observation below. Under markedly wider priors, the posterior distributions for the timing and rate parameters (τ , δ , λ , ϕ) become multimodal. We do not interpret this as evidence that the informative priors used in the main analysis are unbiased; rather, it indicates weak *practical* identifiability of the timing and rate parameters from a single, comparatively short recording under a vague prior, and is precisely the motivation for adopting informative, event-informed priors for this class of model (Section 4.2.1 of the main text) rather than a demonstration that our chosen priors introduce no bias of their own. To make this comparison quantitative rather than a purely visual “overlapping intervals” judgment, we additionally report, for each parameter and each pair of prior specifications: (i) the shift in posterior median expressed in units of the original prior’s standard deviation, and (ii) a distributional overlap coefficient (the overlapping-area statistic between the two posterior densities). All priors, including those induced on the constrained and data-derived-scaled parameters by the transformations in Section 4.2 of the main text, are shown on their natural, interpretable scales in Supplementary Figure S2, rather than only on the unconstrained (logit/log) sampling scale. Quantitative indices of prior

sensitivity analysis are shown in Table S1 and S2 comparing the model estimates on their original scale against wider (Table S1) and narrower priors (Table S2).



Supplementary Figure S2: Prior Sensitivity Analysis. Posterior distributions and 95% credible intervals for all model parameters under three different prior specifications, shown on each parameter's natural, interpretable scale. The overlapping credible intervals support the robustness of the primary finding; multimodality under the widest prior specification reflects weak practical identifiability under vague priors rather than bias in the adopted (informative) priors.

Supplementary Table S1: Prior Sensitivity Analysis Against Wide Priors. Posterior distributions and 95% credible intervals for all model parameters comparing the estimates of the original model with default prior widths, minus the wide prior estimates, shown on each parameter's natural, interpretable scale. The overlapping credible intervals support the robustness of the primary finding. pd: probability of direction, which reflects the probability mass above zero on the same direction of the estimate sign; ps: probability of significance, which indicates the probability mass above 0.1 standard deviations of the parameter natural scale.

Parameter	Estimate	Lower bound	Upper bound	pd	ps
λ	-0.23	-1.77	1.27	61.59	57.70
ϕ	-4.56	-22.44	1.68	94.88	90.76
τ	-0.04	-0.22	0.09	72.78	68.72
δ	0.12	-0.07	0.40	89.13	86.06
α_r	1.14	-11.80	14.67	56.51	52.50
β_r	1.92	-13.83	17.29	59.66	55.84
c_r	-0.01	-0.09	0.05	62.51	58.63
α_s	1.75	-30.12	52.17	53.01	49.16
β_s	-2.18	-47.86	29.10	54.19	50.07
c_s	-0.01	-0.39	0.18	54.72	49.15
c_c	0.00	-0.06	1.00	58.14	10.20
w	0.00	-0.04	0.07	51.38	46.84
$\vec{\pi}_{base}$ [VLF]	0.01	-0.11	0.31	57.12	50.36
$\vec{\pi}_{base}$ [LF]	-0.01	-0.22	0.06	60.75	51.99
$\vec{\pi}_{base}$ [HF]	0.00	-0.11	0.07	54.80	49.98
$\vec{\pi}_{pert}$ [VLF]	-0.06	-0.27	0.13	71.99	68.34
$\vec{\pi}_{pert}$ [LF]	0.05	-0.13	0.26	68.28	64.60
$\vec{\pi}_{pert}$ [HF]	0.01	-0.01	0.04	85.32	82.49

Supplementary Table S2: Prior Sensitivity Analysis Against Narrow Priors. Posterior distributions and 95% credible intervals for all model parameters comparing the estimates of the original model with default prior widths, minus the narrow prior estimates, shown on each parameter's natural, interpretable scale. The overlapping credible intervals support the robustness of the primary finding. pd: probability of direction, which reflects the probability mass above zero on the same direction of the estimate sign; ps: probability of significance, which indicates the probability mass above 0.1 standard deviations of the parameter natural scale.

Parameter	Estimate	Lower bound	Upper bound	pd	ps
λ	0.59	-0.43	1.68	87.25	85.00
ϕ	1.97	-0.06	4.33	97.75	96.91
τ	0.02	-0.09	0.15	65.00	61.10
δ	-0.09	-0.24	0.06	88.40	86.44
α_r	0.74	-9.37	11.63	55.51	51.58
β_r	1.93	-12.12	16.30	60.55	56.76
c_r	-0.01	-0.06	0.05	59.22	55.55
α_s	8.43	-15.46	32.92	74.37	71.21
β_s	-6.86	-29.64	15.77	71.72	68.40

Parameter	Estimate	Lower bound	Upper bound	pd	ps
c_s	-0.03	-0.18	0.13	63.45	59.45
c_c	0.03	-0.05	0.13	84.34	81.16
w	0.01	-0.02	0.05	78.00	74.96
$\tilde{\pi}_{hase}$ [VLF]	0.02	-0.08	0.11	63.03	58.90
$\tilde{\pi}_{base}$ [LF]	-0.01	-0.07	0.05	57.65	53.77
$\tilde{\pi}_{base}$ [HF]	-0.01	-0.07	0.05	63.42	59.71
$\tilde{\pi}_{pert}$ [VLF]	0.05	-0.11	0.19	72.62	69.27
$\tilde{\pi}_{pert}$ [LF]	-0.01	-0.16	0.13	57.86	53.83
$\tilde{\pi}_{pert}$ [HF]	-0.03	-0.07	0.01	94.86	93.70

Section S1.3: Computational Environment

All analyses were performed using `R` (version 4.5.0) and the `rstan` interface (version 2.32.7) to `Stan` (version 2.32.2). The MCMC sampler was run with 4 chains for 10,000 iterations each, with the first 5,000 discarded as warmup, resulting in 20,000 posterior samples. Sampling seed(s): 123. Chain initialization: Stan defaults. Sampler control parameters: `adapt_delta` = 0.9, `max_treedepth` = 10. Hardware: Apple M1 chip, 8 cores and 8 GB of RAM. Wall-clock runtime: 140.74 minutes for the empirical model fit and 68.83, 58.01 and 55.01 minutes for simulation scenario one, two and three, respectively.

Supplementary Table S3 reports R-hat, effective sample size (ESS), and Monte Carlo Standard Error (MCSE) for the high-level parameters. We additionally extend this reporting to the latent Gaussian-process and oscillator parameters, which are not otherwise summarized: R-hat/ESS/MCSE for the per-band GP length-scales ($\rho_{gp,1}, \rho_{gp,2}, \rho_{gp,3}$), and the minimum, median, and maximum R-hat and ESS across the full set of individual non-centered deviates ($z_{gp}, z_{\sin}, z_{\cos}$; $3 \times N_{\sin}$ parameters each).

Section S1.4: Quantification of Modeling Approximations

This section reports two numerical checks of approximations made in the model’s construction (Section 4.1.6 of the main text), performed in response to peer review.

Section S1.4.1: Off-Diagonal Covariance Between Spectral Bands

The model treats Σ_S , the covariance matrix of the three band-specific oscillators $S_j(t_i)$, as diagonal, assuming the VLF, LF, and HF oscillators are mutually uncorrelated within a given record. To quantify this approximation, we computed the full, empirical 3×3 covariance matrix of the realized (centered) band signals, $\text{Cov}[S_{\text{VLF}}(t_i), S_{\text{LF}}(t_i), S_{\text{HF}}(t_i)]$, directly from the posterior draws. Analysis of the empirical dataset yields a mean ratio of the largest off-

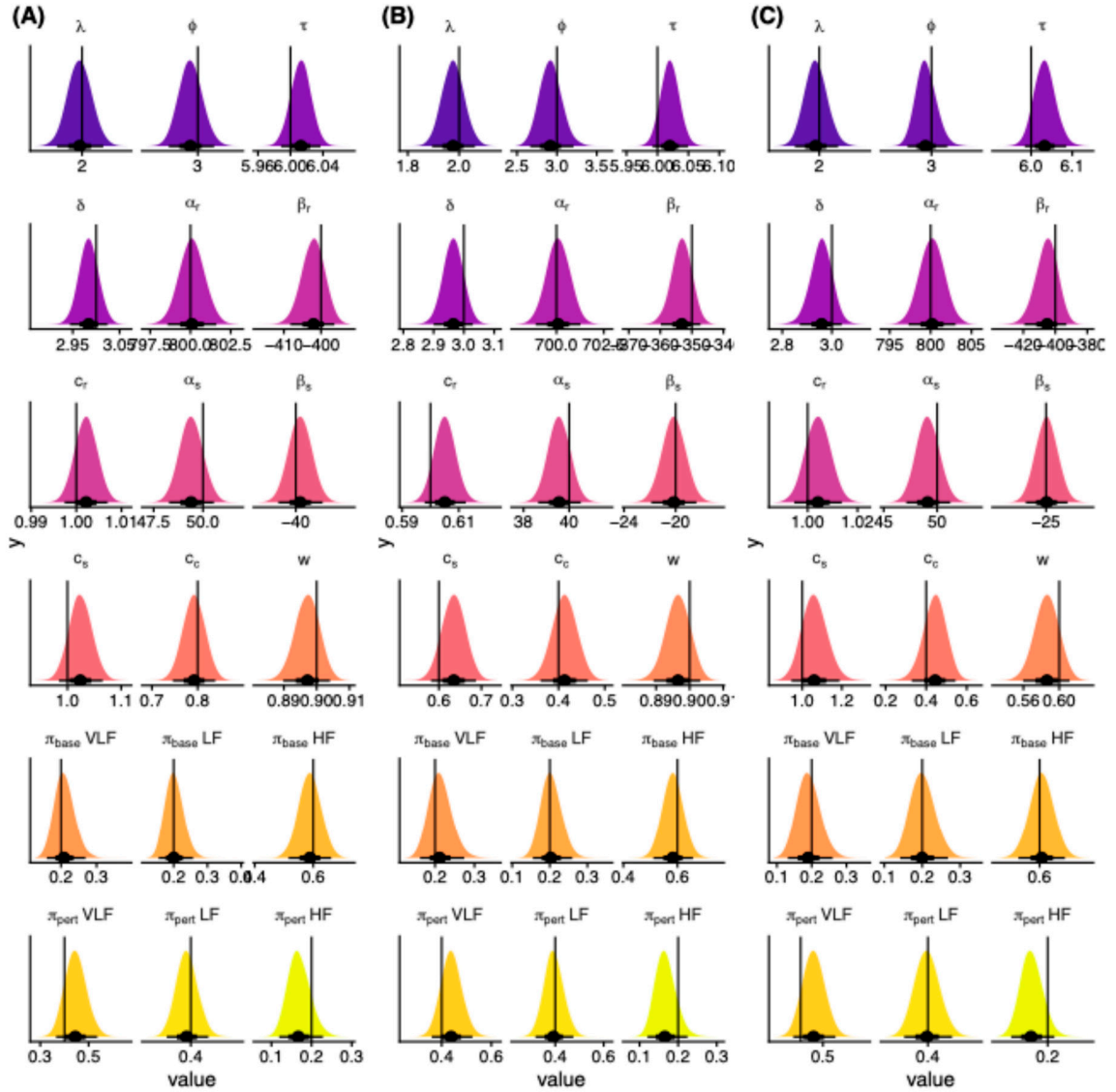
diagonal magnitude to the mean diagonal magnitude of 0.003 (SD 0.001). The 95% credible interval spans from 0.001 to 0.006, with a maximum observed ratio of 0.017 across all iterations. These magnitudes demonstrate that the off-diagonal covariance is negligible, supporting the diagonal approximation within this framework.

Section S2: Extended Simulation Studies

This section provides a complete report of the simulation studies, including full parameter recovery results and an examination of the model's behavior under conditions of misspecification.

Section S2.1: Full Parameter Recovery Results

Supplementary Figure S3 displays the complete posterior distributions for all generative parameters from the three simulation scenarios described in the main text, plotted against their known ground-truth values (red vertical lines). The posteriors are well-centered on the true values, and the true values are consistently contained within the 95% credible intervals, demonstrating parameter recovery.



Supplementary Figure S3: Full Posterior Recovery of Ground-Truth Parameters.

Posterior density plots for all model parameters across the three simulation scenarios (A, B, C). Vertical black lines indicate the true parameter values used to generate each simulated scenario.

We emphasize that this three-scenario recovery exercise is a proof-of-concept parameter-recovery demonstration: truth falling inside the 95% credible interval in three hand-selected cases does not, by itself, constitute a coverage analysis. Sections S2.1.1 and S2.1.2 below report a posterior-correlation analysis and a dedicated simulation-based calibration study addressing this directly.

Section S2.1.1: Posterior Parameter Correlations

To characterize potential trade-offs between parameters, particularly given that the timing/rate parameters (δ , ϕ) are shared between the RR and SDNN trajectories, we report the pairwise posterior correlation structure among β_r , β_s , c_r , c_s , δ , and ϕ , computed from the existing posterior draws of the empirical fit and each simulation scenario. Correlation pairs among the posterior distribution of model parameters across conditions and the empirical case is shown in Supplementary Table S3.

Supplementary Table S3: Posterior Parameter Correlations. Gaussian rank correlations between model parameters within conditions: empirical case and three simulated scenarios. Data is displayed as correlation estimate and 95% confidence interval.

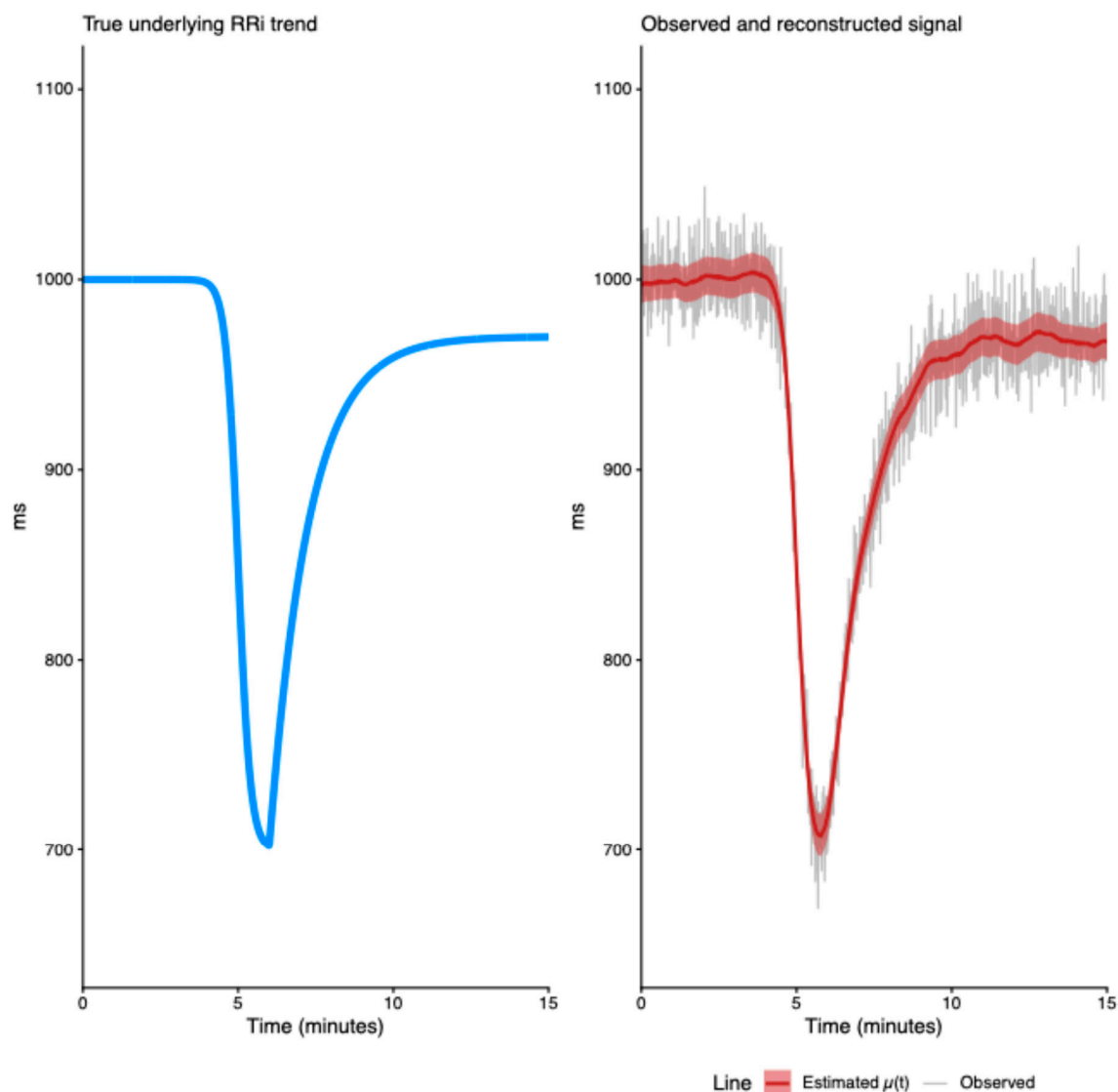
Parameter	Empirical case	Scenario (1)	Scenario (2)	Scenario (3)
$\alpha_r \sim \alpha_s$	0.3 [0.29, 0.32]	0.06 [0.05, 0.08]	0.02 [0, 0.03]	0.05 [0.03, 0.06]
$\alpha_r \sim \beta_r$	-0.73 [-0.74, -0.72]	-0.45 [-0.46, -0.44]	-0.32 [-0.34, -0.31]	-0.4 [-0.41, -0.38]
$\alpha_r \sim \beta_s$	-0.31 [-0.32, -0.3]	-0.12 [-0.13, -0.1]	-0.05 [-0.06, -0.03]	-0.07 [-0.09, -0.06]
$\alpha_r \sim c_r$	-0.01 [-0.02, 0]	-0.02 [-0.03, 0]	0.02 [0, 0.03]	0.02 [0.01, 0.04]
$\alpha_r \sim c_s$	-0.3 [-0.31, -0.28]	-0.69 [-0.7, -0.68]	-0.11 [-0.13, -0.1]	-0.69 [-0.7, -0.68]
$\alpha_r \sim \beta_s$	-0.33 [-0.34, -0.32]	-0.02 [-0.03, -0.01]	0.04 [0.02, 0.05]	-0.02 [-0.04, -0.01]
$\alpha_s \sim \beta_s$	-1 [-1, -1]	-0.75 [-0.76, -0.75]	-0.68 [-0.69, -0.67]	-0.7 [-0.71, -0.7]
$\alpha_s \sim c_r$	0 [-0.02, 0.01]	-0.03 [-0.05, -0.02]	-0.05 [-0.06, -0.03]	-0.01 [-0.03, 0]
$\alpha_s \sim c_s$	-0.36 [-0.37, -0.35]	-0.7 [-0.71, -0.69]	-0.42 [-0.43, -0.41]	-0.71 [-0.71, -0.7]
$\beta_r \sim \alpha_s$	0.31 [0.29, 0.32]	-0.03 [-0.05, -0.02]	0 [-0.02, 0.01]	-0.02 [-0.03, 0]
$\beta_r \sim \beta_s$	-0.28 [-0.29, -0.27]	0.34 [0.32, 0.35]	0.24 [0.23, 0.25]	0.22 [0.21, 0.23]
$\beta_r \sim c_r$	0.02 [0, 0.03]	-0.03 [-0.04, -0.01]	-0.12 [-0.14, -0.11]	-0.1 [-0.12, -0.09]
$\beta_r \sim c_s$	-0.27 [-0.29, -0.26]	0.23 [0.21, 0.24]	-0.84 [-0.84, -0.83]	0.21 [0.2, 0.22]
$\beta_s \sim c_r$	0.22 [0.2, 0.23]	0.01 [-0.01, 0.02]	-0.14 [-0.16, -0.13]	0.02 [0.01, 0.03]
$\beta_s \sim c_s$	0.01 [-0.01, 0.02]	0.07 [0.06, 0.08]	0.14 [0.13, 0.16]	0.08 [0.06, 0.09]
$\beta_s \sim \alpha_s$	0.38 [0.37, 0.39]	0.48 [0.47, 0.49]	-0.12 [-0.14, -0.11]	0.48 [0.47, 0.49]
$\delta \sim \alpha_r$	0.36 [0.35, 0.37]	0.11 [0.1, 0.12]	0.06 [0.05, 0.08]	0.03 [0.01, 0.04]
$\delta \sim \alpha_s$	0.33 [0.32, 0.35]	0.04 [0.02, 0.05]	0.01 [0, 0.02]	0.01 [0, 0.03]
$\delta \sim \beta_r$	0.05 [0.03, 0.06]	0.73 [0.72, 0.73]	0.86 [0.85, 0.86]	0.83 [0.82, 0.83]
$\delta \sim \beta_s$	-0.35 [-0.36, -0.33]	0.23 [0.22, 0.25]	0.21 [0.2, 0.23]	0.18 [0.16, 0.19]
$\delta \sim c_r$	0.01 [0, 0.02]	-0.04 [-0.05, -0.03]	-0.11 [-0.13, -0.1]	-0.09 [-0.11, -0.08]
$\delta \sim c_s$	-0.55 [-0.56, -0.54]	-0.01 [-0.02, 0]	-0.8 [-0.81, -0.8]	0.04 [0.03, 0.05]
$\delta \sim \alpha_s$	-0.06 [-0.08, -0.05]	0.01 [-0.01, 0.02]	-0.11 [-0.13, -0.1]	0.02 [0, 0.03]
$\lambda \sim \alpha_r$	-0.2 [-0.21, -0.18]	-0.44 [-0.45, -0.42]	-0.41 [-0.42, -0.4]	-0.39 [-0.41, -0.38]
$\lambda \sim \alpha_s$	0.58 [0.57, 0.59]	-0.07 [-0.08, -0.05]	-0.01 [-0.03, 0]	-0.03 [-0.04, -0.02]
$\lambda \sim \beta_r$	0.56 [0.55, 0.57]	0.8 [0.8, 0.81]	0.74 [0.74, 0.75]	0.79 [0.78, 0.79]
$\lambda \sim \beta_s$	-0.53 [-0.54, -0.52]	0.3 [0.29, 0.32]	0.18 [0.17, 0.2]	0.19 [0.18, 0.2]
$\lambda \sim \delta$	-0.24 [-0.26, -0.23]	0.38 [0.37, 0.39]	0.44 [0.43, 0.45]	0.52 [0.51, 0.53]
$\lambda \sim \phi$	-0.03 [-0.05, -0.02]	0.45 [0.44, 0.46]	0.38 [0.36, 0.39]	0.42 [0.41, 0.43]
$\lambda \sim \tau$	0.41 [0.4, 0.42]	-0.44 [-0.45, -0.43]	-0.54 [-0.55, -0.53]	-0.56 [-0.57, -0.55]
$\lambda \sim c_r$	0.01 [0, 0.03]	0.01 [0, 0.03]	-0.08 [-0.09, -0.07]	-0.08 [-0.1, -0.07]
$\lambda \sim c_s$	-0.03 [-0.04, -0.01]	0.18 [0.17, 0.19]	-0.59 [-0.6, -0.58]	0.19 [0.17, 0.2]
$\lambda \sim \alpha_s$	0.09 [0.07, 0.1]	0.01 [-0.01, 0.02]	-0.14 [-0.15, -0.12]	0.01 [0, 0.03]
$\phi \sim \alpha_r$	0.13 [0.12, 0.15]	-0.24 [-0.25, -0.22]	-0.17 [-0.18, -0.16]	-0.16 [-0.18, -0.15]
$\phi \sim \alpha_s$	-0.14 [-0.16, -0.13]	-0.05 [-0.07, -0.04]	-0.04 [-0.05, -0.03]	-0.03 [-0.05, -0.02]
$\phi \sim \beta_r$	-0.22 [-0.23, -0.21]	0.67 [0.67, 0.68]	0.61 [0.6, 0.62]	0.64 [0.63, 0.65]
$\phi \sim \beta_s$	0.17 [0.15, 0.18]	0.27 [0.25, 0.28]	0.2 [0.19, 0.21]	0.19 [0.17, 0.2]
$\phi \sim \delta$	-0.52 [-0.53, -0.51]	0.33 [0.31, 0.34]	0.41 [0.4, 0.43]	0.41 [0.4, 0.42]
$\phi \sim \tau$	0.06 [0.04, 0.07]	-0.38 [-0.39, -0.37]	-0.38 [-0.39, -0.37]	-0.44 [-0.45, -0.43]
$\phi \sim c_r$	-0.02 [-0.04, -0.01]	-0.04 [-0.06, -0.03]	-0.09 [-0.1, -0.08]	-0.08 [-0.09, -0.07]
$\phi \sim c_s$	0.03 [0.01, 0.04]	-0.05 [-0.06, -0.03]	-0.62 [-0.63, -0.61]	-0.1 [-0.12, -0.09]

Parameter	Empirical case	Scenario (1)	Scenario (2)	Scenario (3)
$\phi \sim c_s$	-0.13 [-0.14, -0.11]	-0.01 [-0.02, 0.01]	-0.1 [-0.12, -0.09]	0.01 [-0.01, 0.02]
$\tau \sim \alpha_r$	-0.59 [-0.6, -0.58]	-0.14 [-0.16, -0.13]	-0.09 [-0.11, -0.08]	-0.07 [-0.08, -0.06]
$\tau \sim \alpha_s$	-0.34 [-0.35, -0.33]	-0.04 [-0.06, -0.03]	-0.02 [-0.04, -0.01]	-0.02 [-0.04, -0.01]
$\tau \sim \beta_r$	0.18 [0.16, 0.19]	-0.72 [-0.73, -0.72]	-0.84 [-0.84, -0.83]	-0.8 [-0.8, -0.79]
$\tau \sim \beta_s$	0.35 [0.34, 0.36]	-0.24 [-0.25, -0.22]	-0.2 [-0.22, -0.19]	-0.17 [-0.18, -0.16]
$\tau \sim \delta$	-0.78 [-0.79, -0.78]	-0.94 [-0.94, -0.94]	-0.92 [-0.92, -0.92]	-0.93 [-0.93, -0.93]
$\tau \sim c_c$	0.01 [0, 0.03]	0.03 [0.02, 0.05]	0.11 [0.09, 0.12]	0.09 [0.08, 0.1]
$\tau \sim c_r$	0.73 [0.72, 0.74]	0.13 [0.12, 0.15]	0.84 [0.84, 0.84]	0.08 [0.07, 0.1]
$\tau \sim c_s$	0.29 [0.28, 0.31]	0.02 [0.01, 0.04]	0.14 [0.12, 0.15]	0.01 [0, 0.02]
$c_r \sim \alpha_s$	-0.58 [-0.59, -0.57]	-0.03 [-0.04, -0.01]	0 [-0.02, 0.01]	-0.03 [-0.05, -0.02]
$c_r \sim \beta_s$	0.58 [0.57, 0.59]	0.03 [0.01, 0.04]	-0.22 [-0.23, -0.2]	0.03 [0.01, 0.04]
$c_r \sim c_c$	0 [-0.02, 0.01]	0 [-0.01, 0.02]	0.11 [0.09, 0.12]	-0.01 [-0.03, 0]
$c_r \sim c_s$	0.19 [0.18, 0.2]	0.04 [0.03, 0.05]	0.14 [0.12, 0.15]	0.05 [0.03, 0.06]
$c_s \sim c_c$	0.08 [0.07, 0.09]	0.03 [0.01, 0.04]	-0.08 [-0.09, -0.06]	0 [-0.01, 0.02]

Section S2.2: Model Misspecification Analysis

Section S2.2.1: Asymmetric recovery dynamics

We generated a synthetic dataset where the recovery phase followed an exponential decay (a highly asymmetric form) rather than a logistic curve. Supplementary Figure S4 shows the model's attempt to fit this signal. Contrary to first expected, the model captures the general trend and its functional form fully captures the sharp “knee” and long tail of the exponential decay, managing to adapt to this type of model misspecification during the recovery phase, even in the face of a different data generating processes. This provides additional evidence of the current model architecture's flexibility. We extended the residual diagnostics for this check to include formal statistical testing. Residual analysis indicates the discrepancy between the exponential data generation process and the fitted double-logistic curve introduces minimal structural bias into the error term. We evaluated serial independence using the Ljung-Box test, which yielded a non-significant test statistic ($\chi^2(10) = 10.456$, p-value = 0.4015). We assessed the normality of the error distribution via the Shapiro-Wilk test, producing a W-statistic of 0.9984 (p-value = 0.331).



Supplementary Figure S4: Model Fit to Asymmetric Recovery Data. The left panel shows the true underlying heart trajectory (blue line). Right panel shows the model's posterior trajectory (red line) and 95% highest density interval (shaded area in red) is overlaid on synthetic data with an exponential recovery with gaussian noise (grey line).

The R code and model parameters used to generate the data can be seen in the following code block:

```
# --- 1. Define Helper Functions ---
# Standard Logistic (sigmoid) function
logistic_curve <- function(t, location, rate) {
  1 / (1 + exp(-rate * (t - location)))
}
```

```

}

# Asymmetric exponential recovery curve (shaped like a CDF)
exponential_recovery <- function(t, start_time, rate) {
  # Time since the recovery event started
  time_since_start <- pmax(0, t - start_time)
  # Cumulative exponential function, scaled from 0 to 1
  1 - exp(-rate * time_since_start)
}

# --- 2. Set Simulation Parameters ---

# Time parameters
N <- 1200 # Number of data points (e.g., 20 minutes at 1 Hz)
t <- seq(from = 0, to = 15, length.out = N) # Time vector in minutes
tau <- 5 # Time of perturbation event (minutes)
delta <- 1 # Delay between perturbation and recovery onset (minutes)
lambda <- 5 # Rate/steepness of the logistic perturbation
phi <- 0.8 # Rate of the exponential recovery (Lower = slower recovery)

# RRI amplitude parameters
alpha <- 1000 # Initial baseline RRI (ms)
beta <- -300 # Magnitude of the RRI drop (ms)
c_r <- 0.90 # Fractional recovery (0.9 = 90% recovery of the drop)

# Noise parameter
sigma <- 15 # Standard deviation of the residual noise (ms)

# --- 3. Generate the Time Series ---

# Create the perturbation and recovery building blocks
perturbation <- logistic_curve(t, location = tau, rate = lambda)
recovery <- exponential_recovery(t, start_time = tau + delta, rate = phi)

# Combine the components to create the true mean RRI trajectory
# The logic follows the manuscript's model: Baseline - Drop + Recovery
mean_rri <- alpha + (beta * perturbation) - (c_r * beta * recovery)

# Add Gaussian noise to create the final observed signal
set.seed(42) # for reproducibility
observed_rri <- rnorm(n = N, mean = mean_rri, sd = sigma)

# --- 4. Package and Save Data ---

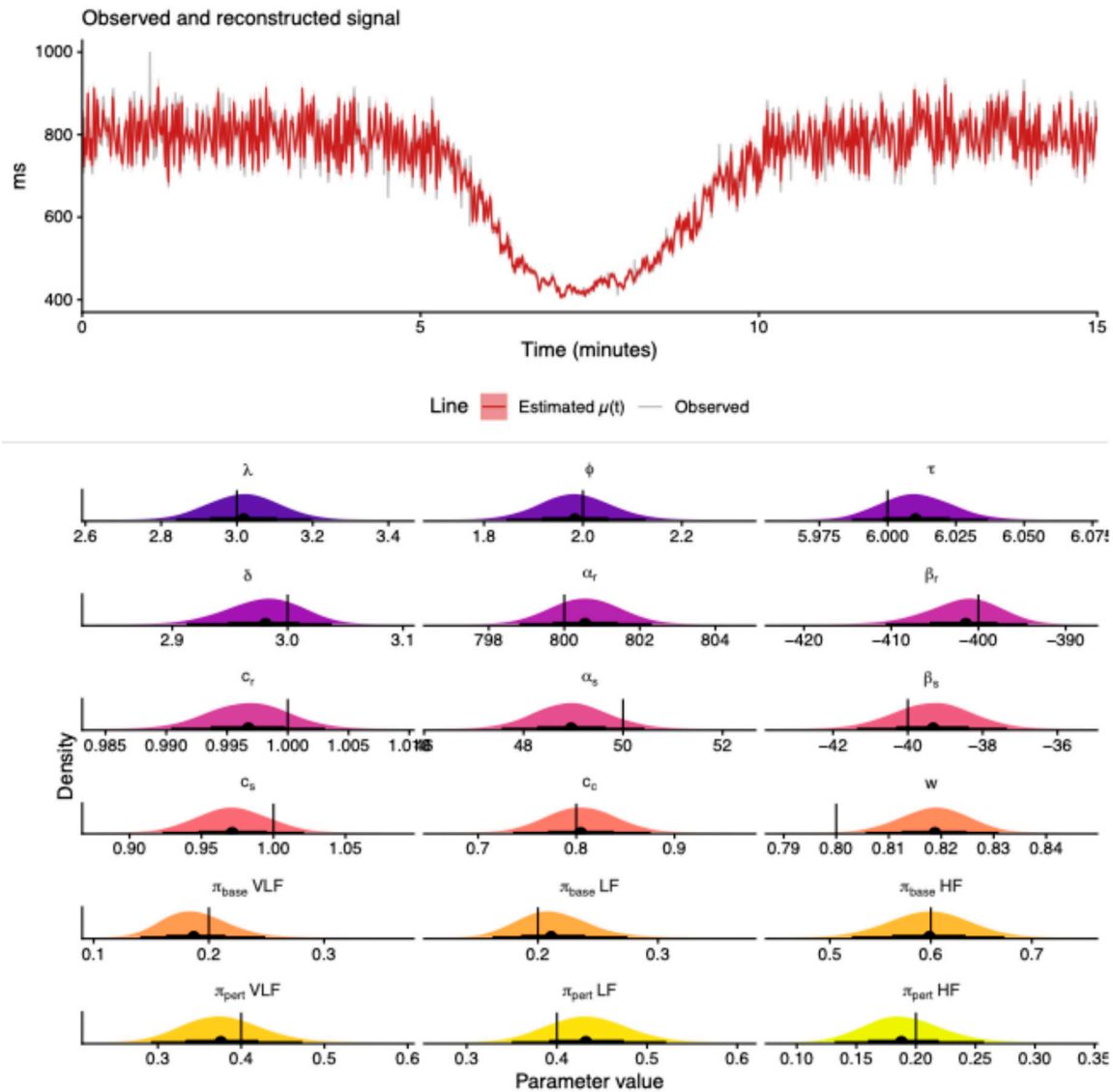
# Store the results in a data frame
sim_data <- data.frame(
  time = t,
  true_mean_rri = mean_rri,
  observed_rri = observed_rri
)

```

Section S2.2.2: Non-Gaussian noise structure

We generated synthetic data using a Student's t distributed noise term with 3 degrees of freedom instead of a Gaussian term. Supplementary Figure S5 demonstrates that the model recovers the primary state trajectories. The posterior estimate for the structured variance fraction (w) exhibits a positive bias, shifting from the ground-truth value of 0.80 to an interval of 0.82 to 0.83. Under a Gaussian likelihood, the model misattributes extreme values from the heavy-tailed noise distribution to the structured oscillatory component.

Despite this likelihood misspecification, the model recovers all remaining dynamic parameters within their 95% credible intervals. Residual diagnostics reflect the expected discrepancy between the data generating process and the Gaussian observation model. The Shapiro-Wilk test confirms non-normality of the residuals ($W = 0.917$, $p < 0.001$). The Ljung-Box test indicates residual serial correlation ($\chi^2(10) = 28.16$, $p = 0.0017$). This correlation emerges because the spectral basis functions partially fit the large stochastic deviations introduced by the t distribution. Applying this model to highly leptokurtic data produces conservative point estimates for unstructured noise and introduces minor autocorrelation in the residuals, while preserving the accuracy of the underlying state trajectories.



Supplementary Figure S5: Impact of Non-Gaussian Noise. In the upper panel, the estimated signal recovered from posterior predictions (in red) overlaid the generated synthetic signal from a heavy-tailed Student's-t noise (in grey). Lower panel shows the posterior distribution for all model parameters when the model is fit to data with heavy-tailed Student's-t noise with 3 degrees of freedom. The ground truth parameters are well recovered except for structured variance fraction w .

The R code used to generate the data is presented next

```
# 1. --- Simulation Setup ---
# Define shared simulation parameters
SIM_DURATION_MIN <- 15 # Total duration in minutes
```

```

SAMPLING_RATE_HZ <- 2 # Sampling rate for RRI series
N_points <- SIM_DURATION_MIN * 60 * SAMPLING_RATE_HZ
time_vector <- seq(0, SIM_DURATION_MIN, length.out = N_points)

# Define frequency bands (VLF, LF, HF)
N_SINUSOIDS <- 20 # Number of sinusoids to approximate spectrum in each band
freq_bands <- list(
  vlf = seq(0.003, 0.039, length.out = N_SINUSOIDS),
  lf = seq(0.040, 0.149, length.out = N_SINUSOIDS),
  hf = seq(0.150, 0.400, length.out = N_SINUSOIDS)
)

# --- Define Parameters for the Three Scenarios ---
# Scenario 1: Classic Sympatho-Vagal Response
# A sharp drop in RR/SDNN with a partial recovery, accompanied by a shift
# from high-frequency (HF) to Low-frequency (LF) power and back.
params <- list(
  # Double-Logistic timing
  lambda = 3, phi = 2, tau = 6, delta = 3,
  # RR(t) params
  alpha_r = 800, beta_r = -400, c_r = 1.0,
  # SDNN(t) params
  alpha_s = 50, beta_s = -40, c_s = 1.0,
  # Spectral & Noise params
  c_c = 0.8, w = 0.80, # 80% structured variance
  # p(t) params
  pi_base = c(0.2, 0.2, 0.6), # VLF, LF, HF - Rest (HF dominant)
  pi_pert = c(0.4, 0.4, 0.2), # VLF, LF, HF - Stress (LF dominant)
  rho_gp = c(1, 1, 1)
)

# -----

# 2. --- Generate and Visualize Data ---
# This block demonstrates how to use the function and visualize the output.

# --- Generate data for Scenario i ---
sim_data <-
  generate_rri_simulation(
    N = N_points,
    t_max = SIM_DURATION_MIN,
    params = params,
    N_sin = N_SINUSOIDS,
    t_dist = TRUE,
    seed = 1,
    nu = 3
  )

```

And the components from the R function `generate_rri_simulation`, used to simulate the data, included the Student-t distribution, is the following:

```

#' Generate a synthetic R-R interval time series from constrained parameters
#'
#' This function encapsulates the full data generating process from the Stan model.
#' ALL model parameters are provided on their constrained, physiologically meaningful
#' scale.
#'
#' @param N Number of data points.
#' @param t_max Maximum time in minutes.
#' @param N_sin Number of sinusoids per frequency band.
#' @param params Named list with model parameters.
#' @param seed An integer for reproducibility.
#' @param ... Currently not used.
#' @param t_dist Logical. Use a t-distribution instead of a normal to generate the data. Default is FALSE.
#' @param nu Numeric. Nu parameter of the Student-t distribution controlling the tails of the distribution.
#' Only used when `t_dist` is TRUE.
#'
#' @return A data frame containing the time vector 't', the final generated
#' 'RR' series, the underlying mean 'mu', and other key components,

```

```

#' including p_vlf/p_lf/p_hf (the mixture/amplitude weights p_j(t); NOT
#' power proportions) and q_vlf/q_lf/q_hf (the implied variance/power
#' share q_j(t), comparable to conventional power-proportion estimates).
generate_rri_simulation <- function(N,
                                   t_max,
                                   params,
                                   N_sin,
                                   seed = 123,
                                   nu = NULL,
                                   t_dist = FALSE) {

  # Helper function to compute the GP's squared exponential covariance kernel
  gp_exp_quad_cov <- function(x, rho) {
    N <- length(x)
    K <- matrix(0, N, N)
    for (i in 1:N) {
      for (j in 1:N) {
        K[i, j] <- exp(-0.5 * ((x[i] - x[j]) / rho)^2)
      }
    }
    return(K)
  }

  # Set seed for complete reproducibility
  set.seed(seed)

  # --- 1. Define Core Dynamic Functions ---
  # These are the double-logistic building blocks for the model's dynamics.
  D_1 <- function(t) { 1 / (1 + exp(-params$lambda * (t - params$tau))) }
  D_2 <- function(t) { 1 / (1 + exp(-params$phi * (t - params$tau - params$delta))) }

  # Define the simulation time grid
  t <- seq(0, t_max, length.out = N)

  # --- 2. Generate Time-Varying Trajectories ---
  # These define the evolution of the signal's mean, total variability, and spectral mix.
  # Sign convention matches the main manuscript / Stan model exactly:
  # mu(t) = alpha + beta*D1(t) - c*beta*D2(t), so a perturbation-induced *drop*
  # requires a negative beta (see the params lists used throughout this project).
  RR_t <- params$alpha_r + params$beta_r * D_1(t) - params$c_r * params$beta_r * D_2(t)
  SDNN_t <- params$alpha_s + params$beta_s * D_1(t) - params$c_s * params$beta_s * D_2(t)
  C_t <- D_1(t) * (1 - params$c_c * D_2(t))

  # Time-varying band mixture/amplitude weights p_j(t) via convex
  # combination (NOT power proportions -- see q_j(t) below)
  p_j <- (1 - C_t) %*% t(params$pi_base) + C_t %*% t(params$pi_pert)
  colnames(p_j) <- c("VLF", "LF", "HF")

  # --- 3. Pre-computation of Basis Functions (like Stan's transformed data) ---
  band_defs <- list(
    VLF = c(0.003, 0.039),
    LF = c(0.040, 0.149),
    HF = c(0.150, 0.400)
  )

  freqs_list <- lapply(band_defs, function(b) seq(b[1], b[2], length.out = N_sin))
  log_freqs_list <- lapply(freqs_list, log)

  # Scale log-frequencies to [0, 1] for a scale-agnostic GP
  log_freqs_scaled_list <- vector("list", 3)
  for (j in 1:3) {
    min_log_f <- min(log_freqs_list[[j]])
    range_log_f <- max(log_freqs_list[[j]]) - min_log_f
    log_freqs_scaled_list[[j]] <- (log_freqs_list[[j]] - min_log_f) / range_log_f
  }

  # Precompute sine/cosine basis matrices and Gram matrices
  sin_mat_list <- vector("list", 3)
  cos_mat_list <- vector("list", 3)
  G_sin_list <- vector("list", 3)
  G_cos_list <- vector("list", 3)
  G_sin_cos_list <- vector("list", 3)

```

```

center_cols <- function(M) sweep(M, 2, colMeans(M), "-")

for (j in 1:3) {
  T_mat <- outer(t * 60, freqs_list[[j]])
  # Center each basis column (sinusoid) BEFORE forming the Gram matrices.
  # This guarantees (a)  $S_j(t) = \sin\_mat \% \% u\_sin + \cos\_mat \% \% u\_cos$  is
  # automatically mean-zero for ANY coefficients, and (b) the Gram
  # matrices below are computed from the exact same centered basis that
  # generates the signal, so  $Var(S_j)$  is computed correctly.
  sin_mat_list[[j]] <- center_cols(sin(2 * pi * T_mat))
  cos_mat_list[[j]] <- center_cols(cos(2 * pi * T_mat))

  normalization <- 1.0 / (N - 1)
  G_sin_list[[j]] <- t(sin_mat_list[[j]]) \% \% sin_mat_list[[j]] * normalization
  G_cos_list[[j]] <- t(cos_mat_list[[j]]) \% \% cos_mat_list[[j]] * normalization
  G_sin_cos_list[[j]] <- t(sin_mat_list[[j]]) \% \% cos_mat_list[[j]] * normalization
}

# --- 4. Generate Spectral Components via GP (like Stan's transformed parameters) ---
# This is the core of the new generative process.

# A. Simulate the standard normal deviates (the "z" parameters)
z_gp <- replicate(3, rnorm(N_sin), simplify = FALSE)
z_sin <- replicate(3, rnorm(N_sin), simplify = FALSE)
z_cos <- replicate(3, rnorm(N_sin), simplify = FALSE)

# B. Loop through bands to generate coefficients
u_sin_list <- vector("list", 3)
u_cos_list <- vector("list", 3)
log_v_list <- vector("list", 3)

for (j in 1:3) {
  # Step 1: Generate the smooth spectral envelope from the GP
  K <- gp_exp_quad_cov(log_freqs_scaled_list[[j]], params$rho_gp[j])
  K <- K + diag(1e-10, N_sin) # Scaled jitter
  L <- t(chol(K)) # Lower triangular Cholesky factor
  log_v <- L \% \% z_gp[[j]]

  log_v_list[[j]] <- log_v

  # Step 2: Normalize the GP output to have unit expected variance
  a_k <- exp(log_v)
  diag_sum <- diag(G_sin_list[[j]]) + diag(G_cos_list[[j]])
  base_v_diag <- sum((a_k^2) * diag_sum)
  full_scale <- a_k / sqrt(base_v_diag + 1e-12)

  # Step 3: Generate the final oscillator coefficients (NCP)
  u_sin_list[[j]] <- z_sin[[j]] * full_scale
  u_cos_list[[j]] <- z_cos[[j]] * full_scale
}

# --- 5. Synthesize Signal and Calculate Final Components ---
S_t_matrix <- matrix(0, nrow = N, ncol = 3)
Sigma_S_diag <- numeric(3) # Will store Var( $S_j$ ) for each band

for (j in 1:3) {
  # Synthesize the j-th oscillator signal using the mean-centered basis;
  #  $S_j$  is therefore exactly mean-centered by construction (matching the Stan
  # implementation's Step 5), and the Gram-matrix-based variance below is exact.
  u_sin <- u_sin_list[[j]]
  u_cos <- u_cos_list[[j]]
  S_j <- sin_mat_list[[j]] \% \% u_sin + cos_mat_list[[j]] \% \% u_cos
  S_t_matrix[, j] <- S_j - mean(S_j) # negligible residual mean kept as a safety check

  # Calculate its *exact* variance using the Gram matrices
  vj <- t(u_sin) \% \% G_sin_list[[j]] \% \% u_sin +
    t(u_cos) \% \% G_cos_list[[j]] \% \% u_cos +
    2 * t(u_sin) \% \% G_sin_cos_list[[j]] \% \% u_cos
  Sigma_S_diag[j] <- vj
}

# Calculate the time-varying scaling amplitude A(t)
var_structured <- SDNN_t^2 * params$w

```

```

variance_terms <- (p_j %%% diag(Sigma_S_diag)) * p_j
denom_sq <- rowSums(variance_terms)
A_t <- sqrt(var_structured) / sqrt(denom_sq + 1e-12)

# Implied variance (power) share per band: q_j(t).
# NOTE: p_j(t) are mixture/amplitude weights, NOT power proportions (see
# manuscript). q_j(t) is the actual proportion of the structured signal's
# variance attributable to band j at time t, and is the correct
# ground-truth target when validating against power-based estimators
# (e.g., STFT/windowed periodogram methods).
q_j <- variance_terms / (denom_sq + 1e-12)
colnames(q_j) <- c("VLF", "LF", "HF")

# Combine components for the final mean trajectory
sum_weighted_S <- rowSums(S_t_matrix * p_j)
mu <- RR_t + A_t * sum_weighted_S

# Generate the final noisy RRI signal
var_noise <- SDNN_t^2 * (1 - params$w)

if (t_dist) {
  sigma_t <- sqrt(var_noise * (nu - 2) / nu)
  # Draw from standardized Student-t and scale/shift
  RRI_t <- mu + sigma_t * rt(N, df = nu)
} else {
  RRI_t <- rnorm(N, mean = mu, sd = sqrt(var_noise))
}

# --- 6. Format Output ---
out <- data.table::data.table(
  t = t,
  RR = RRI_t,
  mu = mu,
  RR_baseline = RR_t,
  SDNN_t = SDNN_t,
  A_t = A_t,
  w = params$w,
  p_vlf = p_j[, 1],
  p_lf = p_j[, 2],
  p_hf = p_j[, 3],
  q_vlf = q_j[, 1],
  q_lf = q_j[, 2],
  q_hf = q_j[, 3]
)

return(list(
  data = out,
  freqs = freqs_list,
  log_v = log_v_list
))
}

```

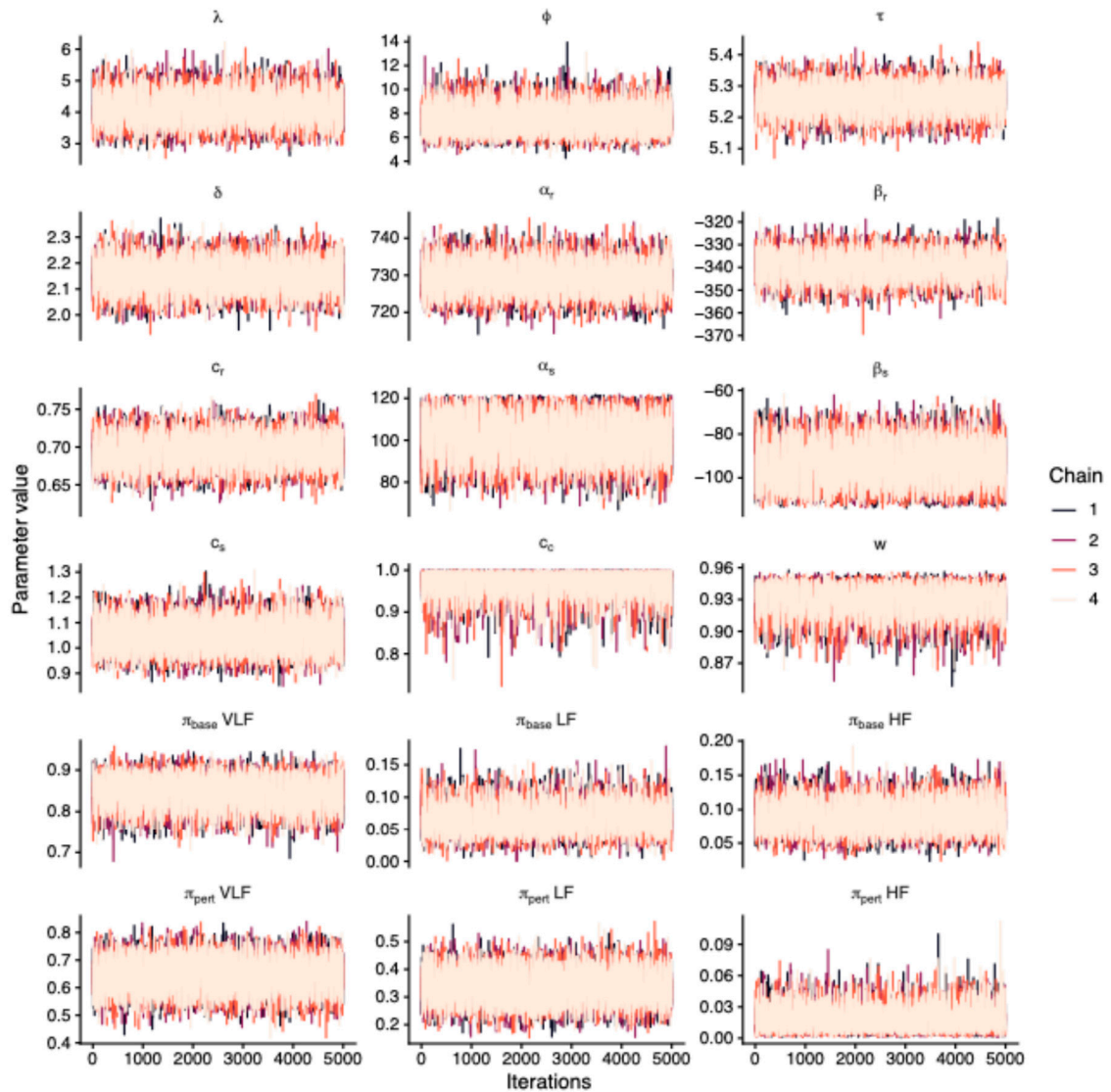
We note that we did not extend this misspecification battery to multiple or overlapping perturbations, non-canonical spectral content, or explicitly non-sinusoidal oscillatory structure. This is a deliberate scope decision, not an oversight: the main text (Section 3.3, Limitations) explicitly states that BAND is designed for a single, isolated perturbation-recovery event and is not intended for naturalistic data containing multiple, overlapping events of unknown form; testing the model against data it is not designed to handle would not be informative about the claims made in this manuscript.

Section S3: Full Empirical Analysis Diagnostics

This section provides a complete set of diagnostic checks for the HMC analysis of the empirical N-of-1 dataset to ensure the reliability of the inference.

Section S3.1: Posterior Distributions

Supplementary Figure S6 shows the traceplots for all key parameters from the empirical model fit, and Figure S7 shows the posterior distribution of model parameters.



Supplementary Figure S6: Full Traceplots for Empirical Analysis. MCMC traceplots for four chains, showing stationary, well-mixed chains.

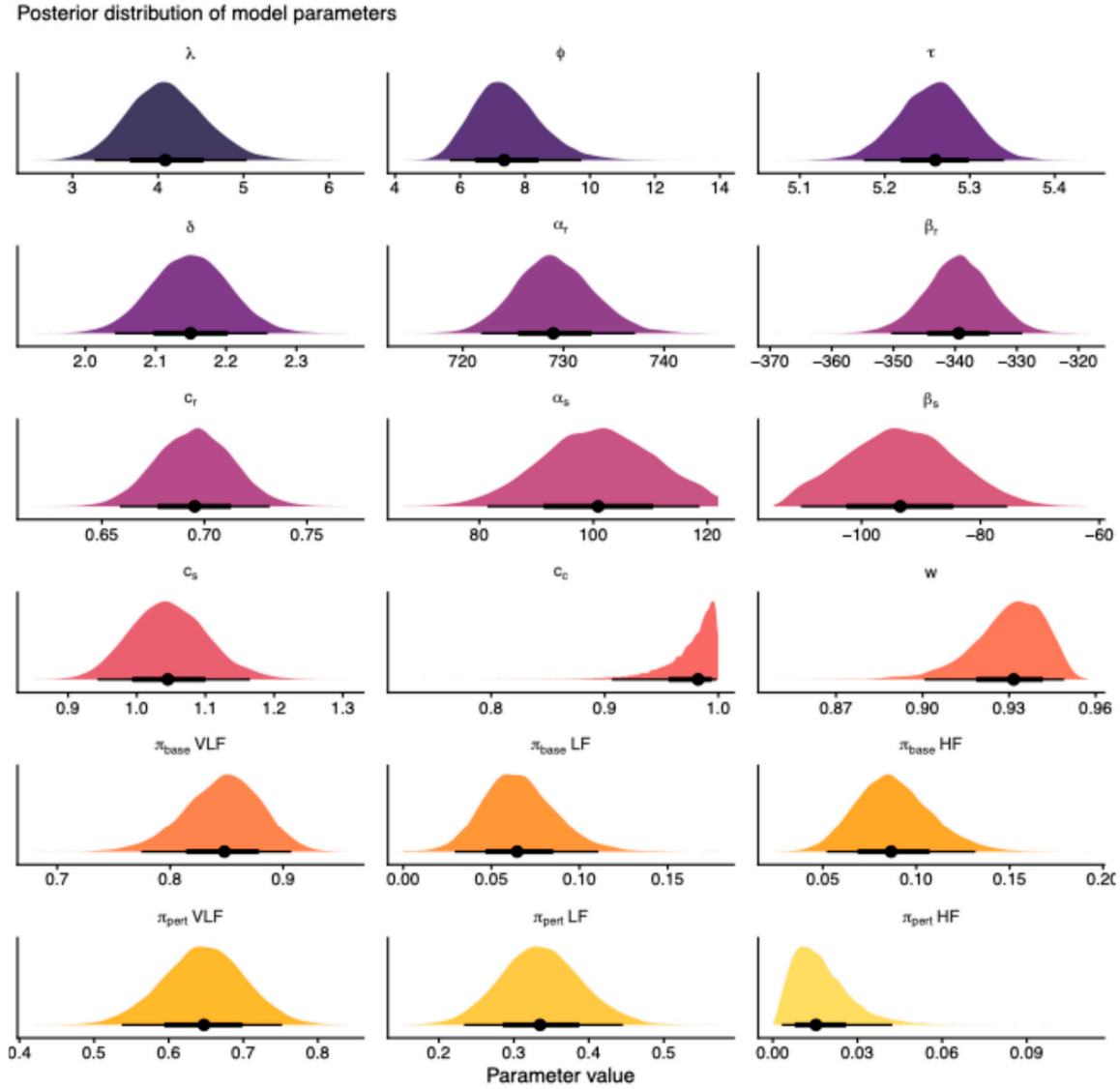


Figure S7. Posterior distribution of phenomenological recovery parameters from the empirical analysis. The posterior distribution for the recovered coefficients of transient changes in mean RRI, total SDNN, and spectral balance. This figure has been regenerated from the single, version-controlled analysis pipeline described in Section S1.3; panel labels and plotted posterior summaries have been cross-checked against the corresponding estimates reported in the main text (e.g., c_s , c_c).

Section S3.2: MCMC Diagnostic Statistics

All MCMC chains converged successfully. There were zero divergent transitions after the warmup period. Supplementary Table S4 provides the R-hat, effective sample size (ESS) and Monte Carlo Standard Error (MCSE) values for all key parameters, confirming the reliability of the posterior estimates.

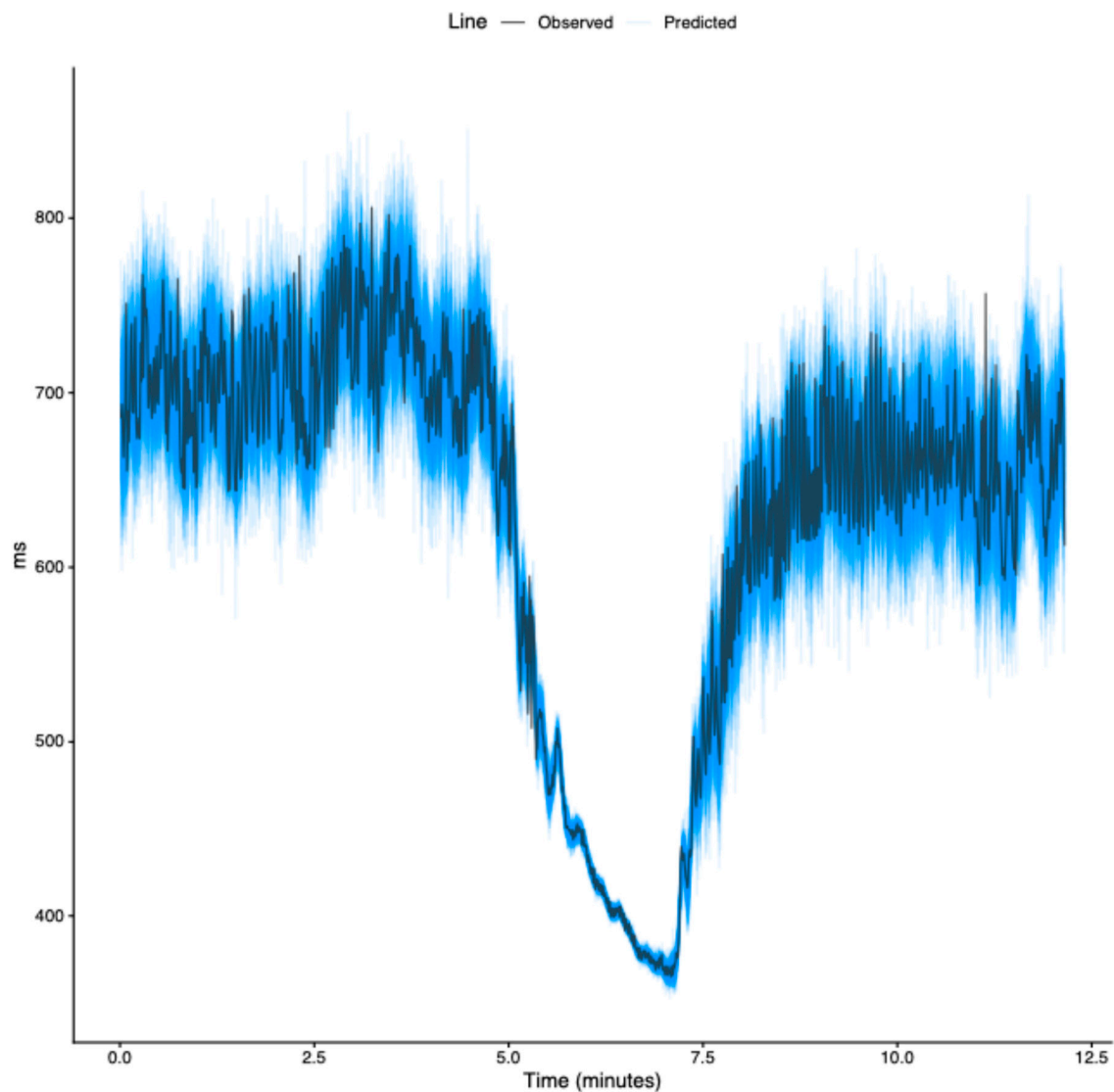
Supplementary Table S4: MCMC Diagnostic Statistics. All R-hat values are near 1.00 up to three digits, both tail and bulk ESS are greater than 5,000, confirming the reliability of the posterior estimates. This table now additionally reports $\rho_{gp,1-3}$ and a min/median/max summary across the individual z_{gp} , z_{sin} , and z_{cos} deviates, per Section S1.3.

Parameter	MCSE Mean	Bulk ESS	Tail ESS	R-hat
λ	0.004	10955.150	11681.879	1.00014
ϕ	0.007	19902.905	14863.888	1.00009
τ	0.000	11726.533	13216.010	1.00006
δ	0.001	10968.645	13003.751	0.99996
α_r	0.035	12242.733	13600.879	1.00015
β_r	0.054	9938.473	12320.797	1.00025
c_r	0.000	9911.757	11813.816	1.00005
α_s	0.106	8337.728	10864.137	0.99998
β_s	0.098	8487.815	11043.453	0.99999
c_s	0.000	22380.014	15972.501	1.00002
c_c	0.000	21578.002	14118.697	0.99998
w	0.000	8129.800	10754.198	1.00000
$\vec{\pi}_{base}$ [VLF]	0.000	6900.453	11503.989	0.99999
$\vec{\pi}_{base}$ [LF]	0.000	6260.435	8407.045	1.00008
$\vec{\pi}_{base}$ [HF]	0.000	7598.213	12219.495	1.00001
$\vec{\pi}_{pert}$ [VLF]	0.001	10779.807	13185.338	1.00029
$\vec{\pi}_{pert}$ [LF]	0.001	10748.618	12751.512	1.00027
$\vec{\pi}_{pert}$ [HF]	0.000	16961.125	11688.617	0.99997
ρ_{gp} [VLF]	0.002	8021.760	9299.197	1.00000
ρ_{gp} [LF]	0.001	15237.119	11895.633	1.00000
ρ_{gp} [HF]	0.002	5409.450	4697.835	1.00009
z_{cos} (Min)	0.003	14421.000	12552.000	0.99995
z_{cos} (Median)	0.004	30867.000	14457.000	1.00002
z_{cos} (Max)	0.005	45588.000	16362.000	1.00104
z_{gp} (Min)	0.005	15598.000	11915.000	0.99995
z_{gp} (Median)	0.005	39110.000	14231.000	1.00002
z_{gp} (Max)	0.008	44209.000	16253.000	1.00125
z_{sin} (Min)	0.003	15663.000	12418.000	0.99995
z_{sin} (Median)	0.004	31138.000	14556.000	1.00003
z_{sin} (Max)	0.005	42072.000	16639.000	1.00067

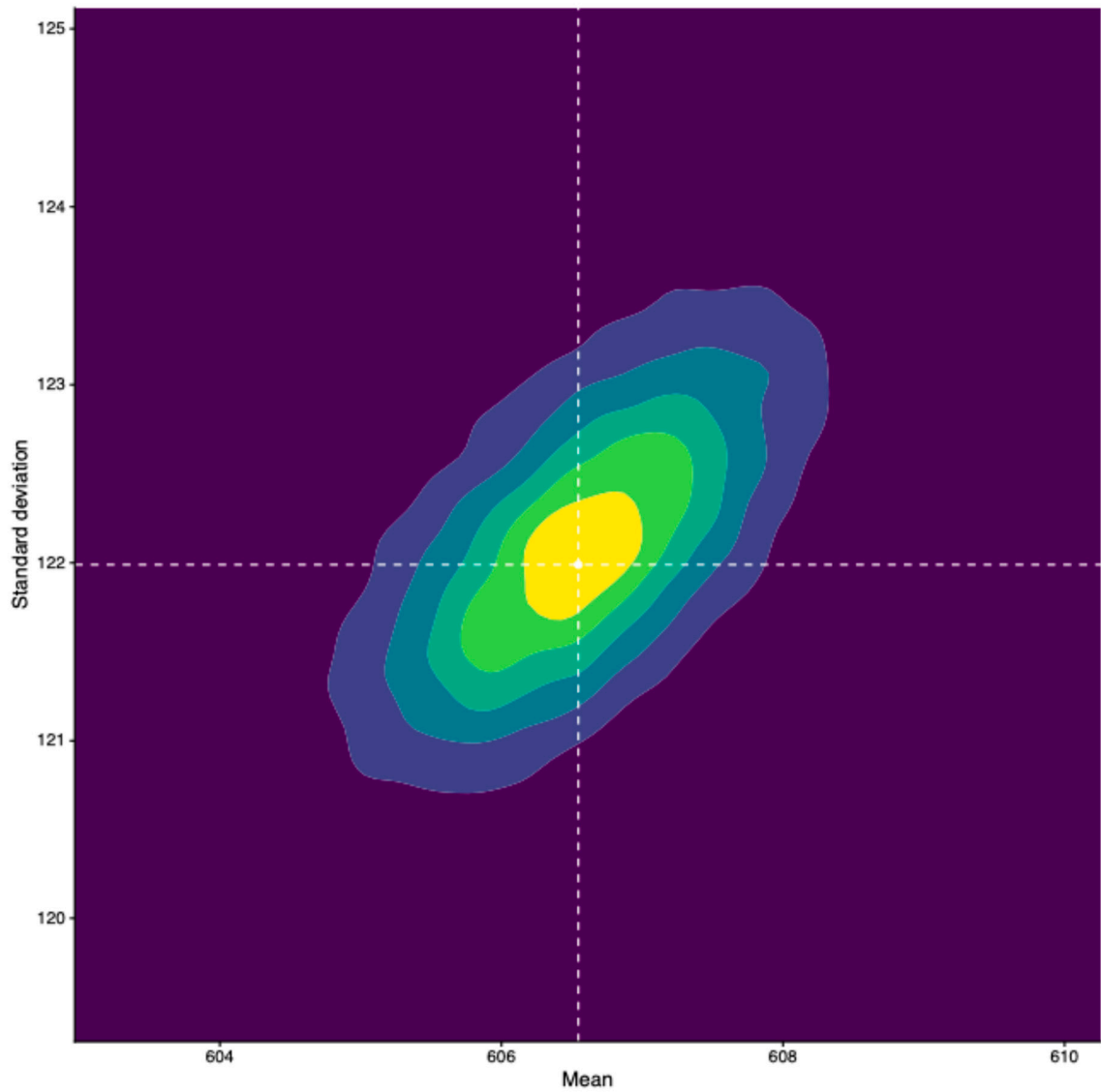
Section S3.3: Posterior Predictive Checks (PPC)

To assess the model's descriptive adequacy, we performed posterior predictive checks. Supplementary Figure S8 overlays the empirical RRI time series with 100 time series simulated from the fitted model's posterior predictive distribution. The simulated data closely

resembles the observed data in its overall structure and variance. Furthermore, Supplementary Figure S9 shows that the distributions of key summary statistics (mean and SD) from the simulated data are well-centered on the values from the observed data, indicating an adequate model fit.



Supplementary Figure S8: Graphical Posterior Predictive Check. The observed RRI data (dark line) is shown with 100 replicated datasets (faint blue lines) generated from the fitted model.



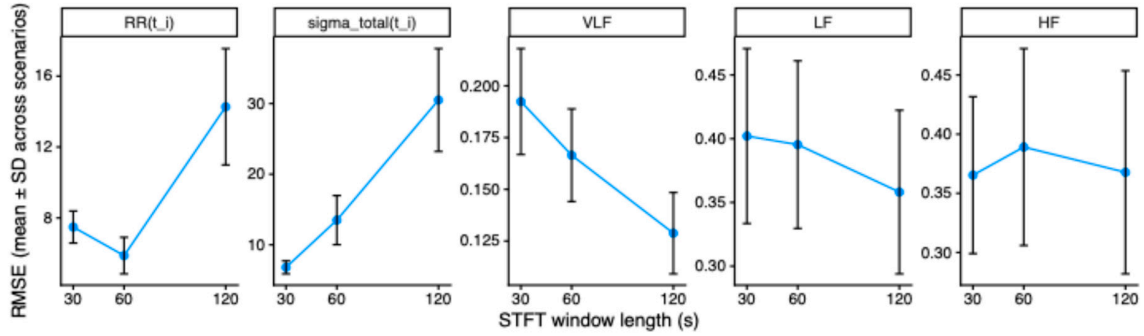
Supplementary Figure S9: Distributional Posterior Predictive Checks. The distribution for two summary statistics calculated from the replicated datasets. The vertical and horizontal white line shows the value of the statistic from the actual observed data, meanwhile the 2-dimensional filled density highlight the distribution of summary statistics from the predicted R_{Ri} signal.

Section S4: Comparator Implementations and Benchmarking

This section documents the conventional and alternative methods used as comparators in Section 2.1.1 and Table 3 of the main text, added and expanded in response to peer review.

Section S4.1: Sliding-Window and Short-Time Fourier Transform (STFT)

Time-domain comparators (mean RRi and SDNN) were computed using a sliding-window mean and standard deviation. Spectral comparators used an STFT with a matched window length. The primary window length was 60 seconds; sensitivity to this choice was assessed by repeating the analysis with 30-second and 120-second windows. The results for different latent states reconstruction can be observed in Supplementary Figure S10, and the companion Supplementary Table S5.



Supplementary Figure S10: RMSE of Sliding window and Short-Time Fourier Transform (STFT) at different windows. The distribution Root mean squared error (RMSE) for each latent state computed from the BAND model calculated from the reference datasets from different length windows.

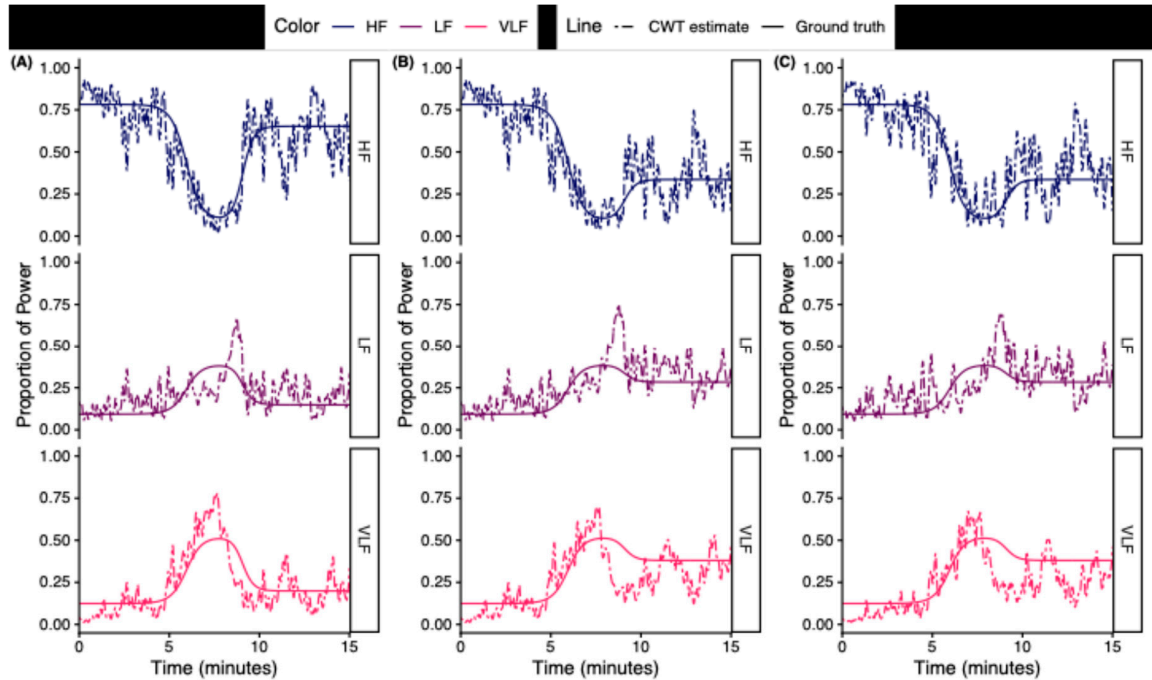
Supplementary Table S5: Sliding window and Short-Time Fourier Transform (STFT) sensitivity analysis. Root mean squared error (RMSE) statistic of each sliding window width configuration against ground truth parameters.

Domain	Window (s)	Mean RMSE (across scenarios)	SD RMSE
VLF	$RR(t_i)$ 30	7.485	0.903
	$RR(t_i)$ 60	5.874	1.031
	$RR(t_i)$ 120	14.271	3.283
	$\sigma_{total}(t_i)$ 30	6.810	0.920
	$\sigma_{total}(t_i)$ 60	13.499	3.478
	$\sigma_{total}(t_i)$ 120	30.503	7.279
LF	30	0.192	0.026

Domain	Window (s)	Mean RMSE (across scenarios)	SD RMSE
VLF	60	0.166	0.022
VLF	120	0.129	0.020
LF	30	0.402	0.069
LF	60	0.395	0.066
LF	120	0.358	0.064
HF	30	0.365	0.066
HF	60	0.389	0.083
HF	120	0.368	0.086

Section S4.2: Continuous Wavelet Transform (CWT)

As a further, more robust spectral comparator, we implemented a continuous wavelet transform (Morlet wavelet) analysis, given that the CWT does not impose a single, fixed time-frequency resolution trade-off across all frequencies, unlike the STFT. Band-wise power proportions (VLF, LF, HF) were extracted from the resulting scalogram by integrating power within each canonical band at each time step and normalizing to yield proportional power, using the same per-time-step normalization applied to the STFT output (Section 4.3.2 of the main text); this is directly comparable to the model's implied variance-share quantity, $q_j(t_i)$ (Section 4.1.3 of the main text), rather than to the raw mixture weight $p_j(t_i)$. The CWT method comparison against the ground truth can be observed in Supplementary Figure S11.



Supplementary Figure S11: Temporal reconstruction of spectral trajectories from Continuous Wavelet Transform (CWT). Band-wise power proportions (VLF, LF, HF) were extracted from the resulting scalogram by integrating power within each canonical band at each time step and normalizing to yield proportional power.