PNPInverse Weekly Development Writeup
Week of February 16, 2026

Jake Weinstein (Project Notes Consolidated)

February 19, 2026

Weekly Focus

This week focused on converting the inverse workflow from isolated scripts into a modular framework, then building a new experiment-style inference path for Robin boundary transfer coefficients from ϕ_{applied}-vs-steady-flux data.
The center of gravity was the new Robin curve inference path: data generation, steady-state detection, adjoint gradient extraction point-by-point, SciPy-based optimization, and recovery logic for forward-solve failures.

Outline of What Was Implemented

1. Unified inverse interface that can plug into compatible forward solvers and infer multiple parameter targets.
2. Always-resilient optimization logic that can recover from nonlinear forward-solve failures.
3. Robin experimental pipeline where the observable is steady-state flux on the Robin boundary.
4. New gradient-based Robin κ inference from flux-curve mismatch using Firedrake adjoints + SciPy.
5. Current-density-proxy inference branch using charge-weighted flux $\sum_i z_i F_i$, with Faraday scaling intentionally omitted pending unit calibration.
6. Regenerated report set for 0%, 2.5%, and 5% noise in both data-generation and inference-fit formats, plus a 2.5% convergence GIF artifact.
7. Initial-condition optimization study showing a measurable runtime reduction by replacing blob IC with a simple near-steady initial condition.
8. Replay diagnostics and fallback/rebuild instrumentation for current-density-proxy inference, followed by temporary replay deactivation due validity concerns.

9. Process-parallel point-solve execution for the 15-point ϕ_{applied} sweep, with per-worker tape isolation and runtime benchmarking.

1. Unified Inverse Interface

What was built

A shared inference core was established so new inverse problems can be configured through:

- a forward-solver adapter,
- a parameter target definition,
- a request object containing true value, initial guess, noise, and optimizer settings.

This unifies diffusion, Dirichlet BC, and Robin BC inference under one architecture while preserving compatibility wrappers for existing helper imports.

Mathematical template

For state $u(m)$ produced by the forward solve under control/parameter m, the generic objective template is:

$$J(m) = \frac{1}{2} \int_{\Omega} \|\mathcal{O}(u(m)) - d\|^2 \, dx,$$

where $\mathcal{O}$ selects the measured field(s) (concentration, potential, or derived observable), and d is synthetic/experimental target data.

2. Resilient Optimization for Forward-Solve Failures

What was built

The optimization loop was made always-resilient by adding staged recovery rather than hard-failing on the first diverged PDE solve. Recovery phases were implemented in this order:

1. Increase `snes_max_it`.
2. Apply anisotropy reduction to the trial parameter vector (with reset of relaxed tolerances).
3. Relax nonlinear/linear tolerances (`snes_atol`, `snes_rtol`, `ksp_rtol`) and line-search strategy.

The restart logic uses the last known feasible parameter state, so retries do not repeatedly jump back to the original initial guess.

Anisotropy reduction concept

For a multi-component parameter vector m, anisotropy is treated as a max/min magnitude ratio. If too large, the vector is blended toward an isotropic surrogate based on geometric-mean magnitude:

$$m_{\text{new}} = (1 - \beta)m + \beta m_{\text{iso}},$$

where $\beta \in [0, 1]$ controls flattening strength.

3. Robin Experimental Flux Pipeline

What was built

A new workflow mirrors an experiment where:

- control input is applied voltage ϕ_{applied},
- measured output is steady-state flux through the Robin boundary.

Utilities were added for:

- steady-state probe sweeps,
- synthetic ϕ_{applied}-flux data generation,
- CSV serialization and reuse across fitting runs.

Steady-state flux definition

The Robin boundary law in this implementation is:

$$J_i \cdot n = \kappa_i(c_i - c_{\infty,i}).$$

Species flux across the Robin electrode boundary is assembled as

$$F_i = \int_{\Gamma_{\text{electrode}}} \kappa_i(c_i - c_{\infty,i})\, ds.$$

The default scalar observable used this week is total species flux:

$$F_{\text{obs}} = \sum_i F_i.$$

From Modeled Flux to Experimental Current Density

Experimentally, the directly measured quantity is typically current density ($\mathrm{mA/cm^2}$), not molar flux. In this workflow, the key relation is:

$$\text{charge-weighted flux } \propto \sum_i z_i F_i,$$

so to move from total species flux to a current-like observable, we weight each species contribution by its charge and sum.

If physical current density units are needed, this weighted sum is then scaled by constants (Faraday factor and boundary-area conversion). For fitting, the important structural step is the charge weighting itself.

Current implementation note (temporary): the new current-density inference branch currently omits the Faraday factor on purpose and fits a charge-weighted flux proxy $\sum_i z_i F_i$ in arbitrary units. This keeps curve magnitudes in the same $\mathcal{O}(1)$ range as the flux-based studies while unit conventions are being finalized. Therefore, the present proxy scaling is not yet a physically calibrated $\mathrm{mA/cm^2}$ mapping.

Steady-state criterion

At each time step n, with per-species boundary flux $F_i^{(n)}$:

$$\Delta_i^{(n)} = |F_i^{(n)} - F_i^{(n-1)}|,$$

$$\mathrm{rel}^{(n)} = \max_i \frac{\Delta_i^{(n)}}{\max(|F_i^{(n)}|, |F_i^{(n-1)}|, \varepsilon_{\mathrm{abs}})}, \quad \mathrm{abs}^{(n)} = \max_i \Delta_i^{(n)}.$$

A step is steady if

$$\mathrm{rel}^{(n)} \le \varepsilon_{\mathrm{rel}} \quad \text{or} \quad \mathrm{abs}^{(n)} \le \varepsilon_{\mathrm{abs}},$$

and steady state is declared after the required number of consecutive steady steps.

How κ Shapes the ϕ_{applied} Curves

To isolate κ-dependence, a no-noise overlay was generated for:

$$\kappa \in \{[2,2],[1,1],[2,1],[1,2],[1,5],[5,1]\}.$$

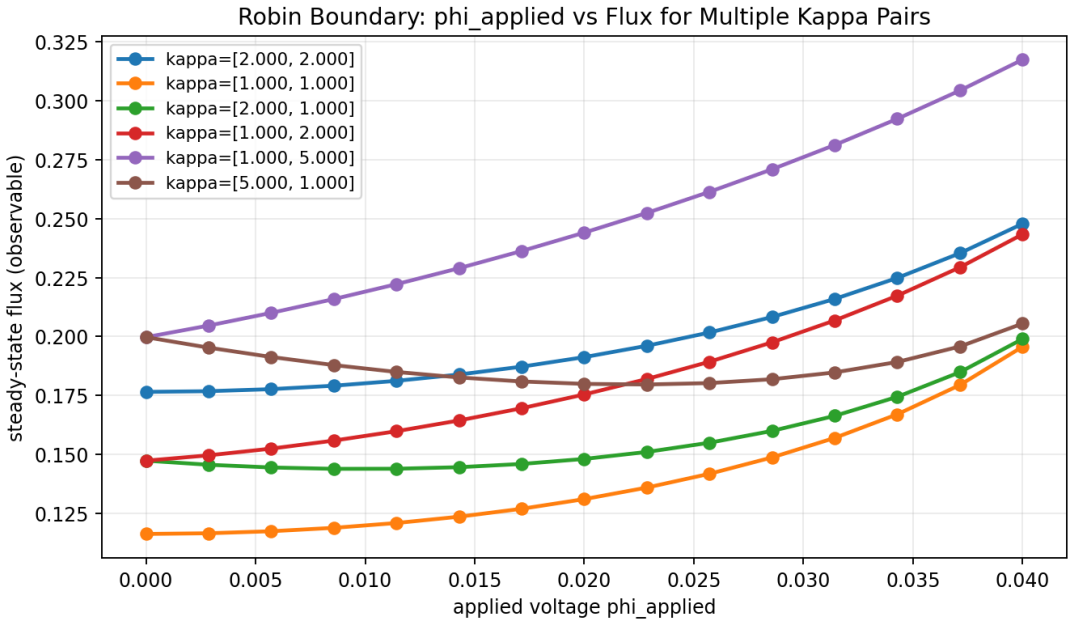

Figure 1: No-noise ϕ_{applied}-vs-steady-flux curves for multiple Robin κ pairs (original Figure 1).

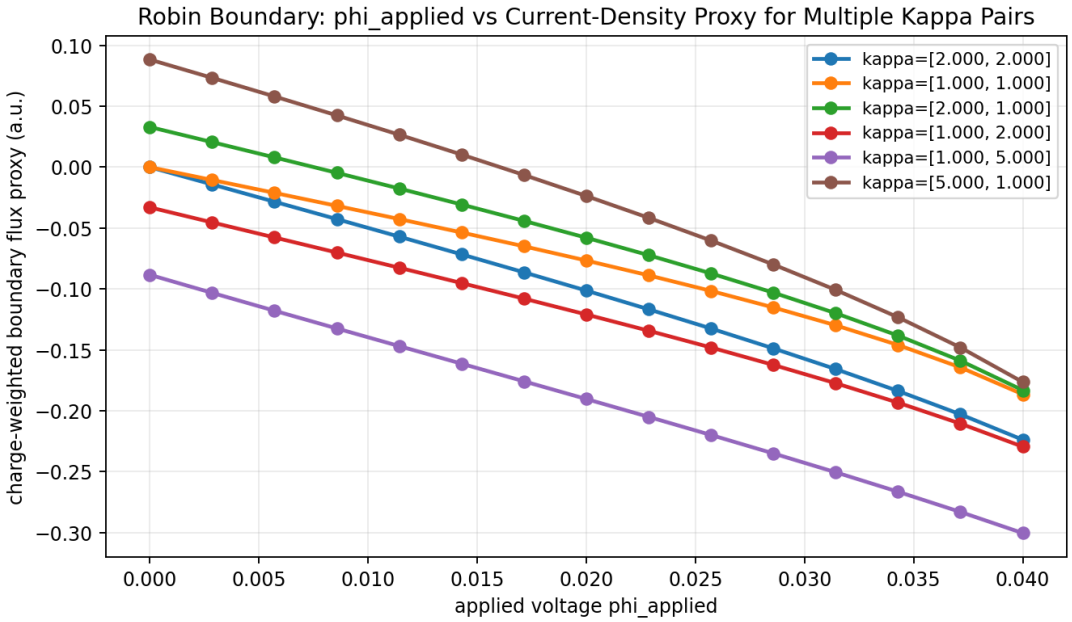

Figure 2: No-noise ϕ_{applied}-vs-current-density-proxy curves for the same Robin κ pairs (added below Figure 1).

Interpretation from the current-density-proxy overlay (second figure):

- In this voltage window, all curves have negative slope: increasing ϕ_{applied} drives the charge-weighted proxy to more negative values.

- Zero-voltage intercepts depend strongly on anisotropy: $[1,5]$ and $[1,2]$ start negative, $[5,1]$ and $[2,1]$ start positive, while $[1,1]$ and $[2,2]$ start near zero.
- Changing which species gets the larger κ ($[1,5]$ vs $[5,1]$) changes both intercept and slope, consistent with asymmetric species contributions to the charge-weighted observable.

Using endpoint slope $\big(P(0.04) - P(0)\big)/0.04$ for proxy $P = \sum_i z_i F_i$, representative values are:

$$[1,1]: -4.661, \quad [2,2]: -5.594, \quad [2,1]: -5.402, \quad [1,2]: -4.911, \quad [1,5]: -5.301, \quad [5,1]: -6.619.$$

4. Robin κ Inference from Flux Curves

What was built

A dedicated script and helper pipeline were implemented for inferring κ from a full ϕ_{applied}-vs-steady-flux curve.
Per voltage point ϕ_j, a steady-state forward solve is run and compared to target flux $F_j^\star$.

Objective and gradient

Per-point loss:

$$L_j(\kappa) = \frac{1}{2}\left(F_j(\kappa) - F_j^\star\right)^2.$$

Global loss over m sweep points:

$$J(\kappa) = \sum_{j=1}^{m} L_j(\kappa).$$

Firedrake adjoint is used at each point to compute

$$\nabla_\kappa L_j(\kappa),$$

and gradients are accumulated over converged points:

$$\nabla J(\kappa) = \sum_{j \in \mathcal{C}} \nabla_\kappa L_j(\kappa).$$

SciPy minimize pipeline

SciPy receives:

- `fun`: curve loss $J(\kappa)$,
- `jac`: analytic adjoint gradient $\nabla J(\kappa)$,
- bounds on κ,
- method/tolerance/options (e.g. `L-BFGS-B`, `ftol`, `gtol`, `maxiter`),

- callback for iteration diagnostics and plot updates.

5. Noise Model Clarification

Noise is Gaussian with standard deviation tied to RMS signal magnitude:

$$\sigma = \left(\frac{p}{100}\right) \mathrm{RMS}(F_{\mathrm{clean}}),$$

where p is `noise_percent`.
So 5% means $\sigma = 0.05 \cdot \mathrm{RMS}$, not a hard $\pm 5\%$ pointwise cap. Individual points can exceed 5% in magnitude.

6. Representative Results and Plots

Summary Tables

Case	Seed	Mean $\lvert\Delta F/F\rvert$ (%)	Max $\lvert\Delta F/F\rvert$ (%)	Best κ_0	Best κ_1	Final Loss
0% noise	20260220	0.000	0.000	1.0015	1.9980	1.4901×10^{-8}
2.5% noise	20260222	1.790	8.263	1.0165	1.9927	1.6312×10^{-4}
5% noise	20260221	3.908	7.918	1.4533	1.7969	4.0573×10^{-4}

Case	Target κ_0	Target κ_1	Fit κ_0	Fit κ_1	RMSE (Flux)	MAE (Flux)
0% noise	1.0000	2.0000	1.0015	1.9980	4.4573×10^{-5}	3.8524×10^{-5}
2.5% noise	1.0000	2.0000	1.0165	1.9927	4.6636×10^{-3}	3.0610×10^{-3}
5% noise	1.0000	2.0000	1.4533	1.7969	7.3551×10^{-3}	5.3404×10^{-3}

Case	Objective Evals	SciPy Success	Termination
0% noise	20	True	Proj. grad $\leq$ PGTOL
2.5% noise	20	True	Proj. grad $\leq$ PGTOL
5% noise	16	True	Proj. grad $\leq$ PGTOL

Data Generation Curves

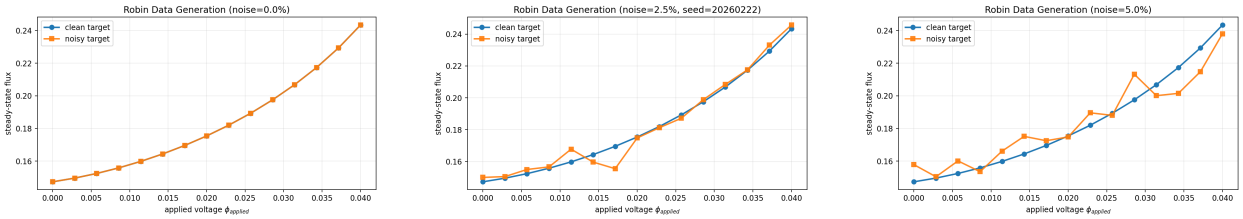

Figure 3: Synthetic Robin flux-curve data generation ordered from least to most noise: 0%, 2.5%, 5%.

Inference Fits

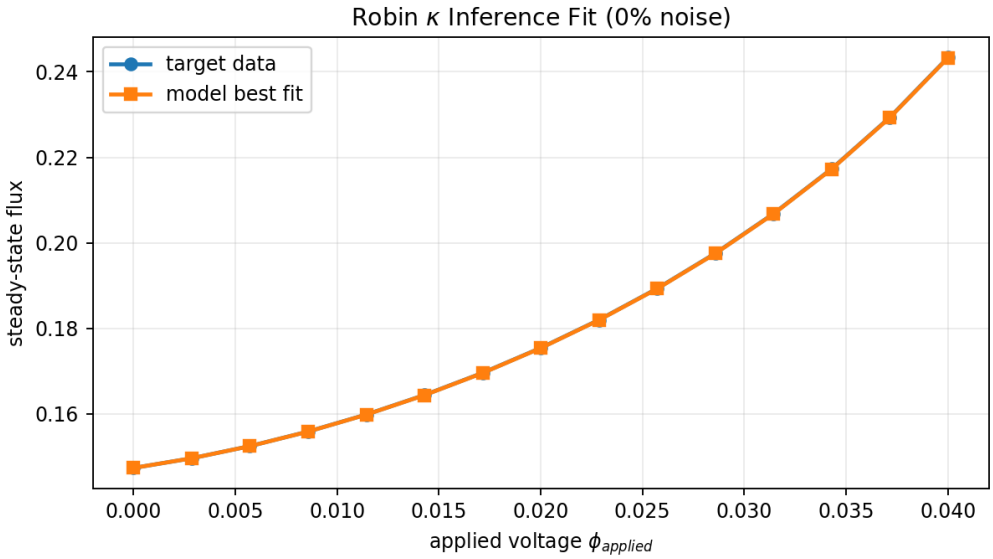

Figure 4: Robin κ curve fit at 0% noise.

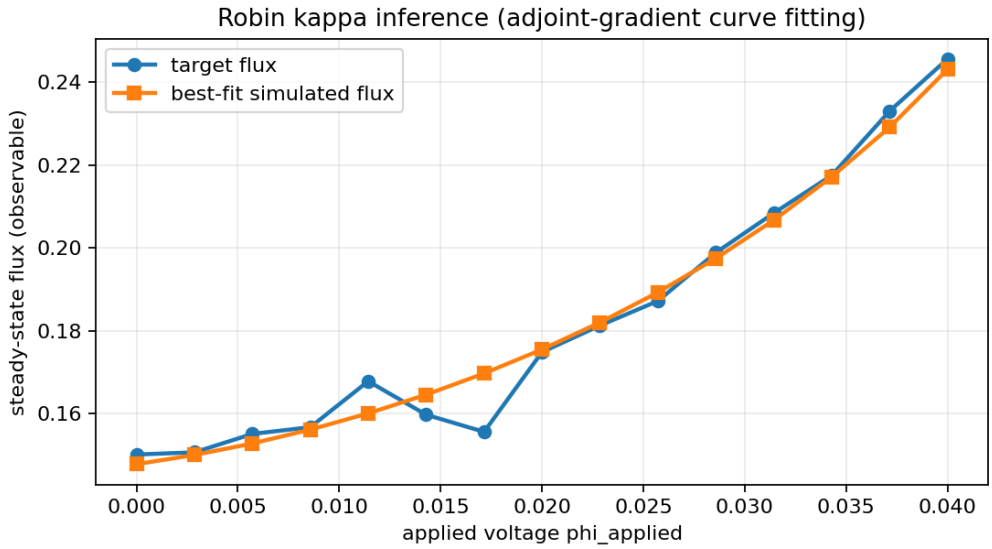

Figure 5: Robin κ curve fit at 2.5% noise.

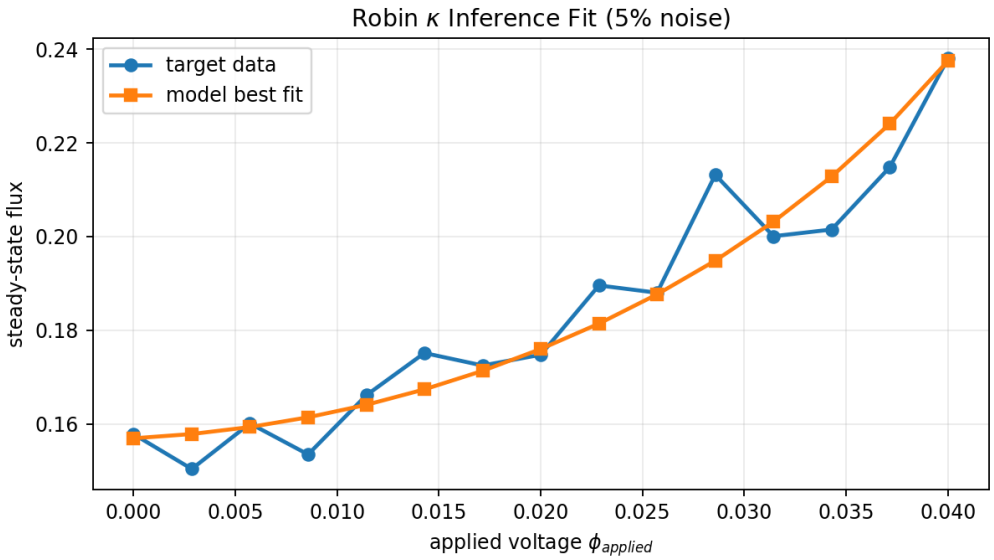

Figure 6: Robin κ curve fit at 5% noise.

Current-Density Counterparts (Proxy Scaling)

An analogous set of plots was generated for the current-density inference path. In this revision, the observable is the charge-weighted flux proxy $\sum_i z_i F_i$ (Faraday scaling intentionally omitted), so the y-axis is in arbitrary units. The objective and optimizer are unchanged.

Case	Seed	Best κ_0	Best κ_1	Final Loss
0% noise	20260220	0.9989	1.9982	1.1347×10^{-8}
2.5% noise	20260222	1.0064	2.0014	9.0801×10^{-5}
5% noise	20260221	1.9494	2.6694	5.1787×10^{-4}

Table 1: Current-density-proxy inference summary (arbitrary units, no Faraday scaling).

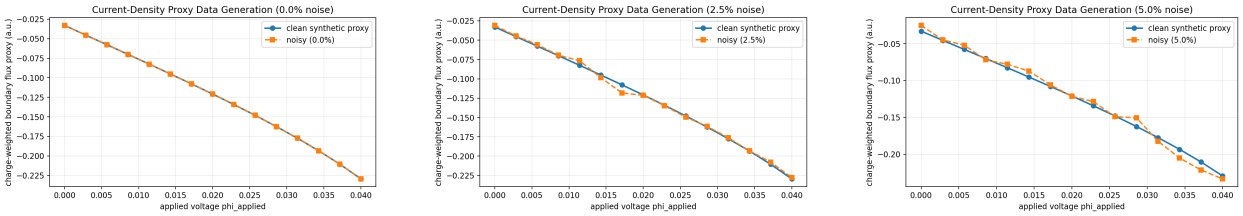

Figure 7: Synthetic current-density-proxy data generation (0%, 2.5%, 5% noise; arbitrary units).

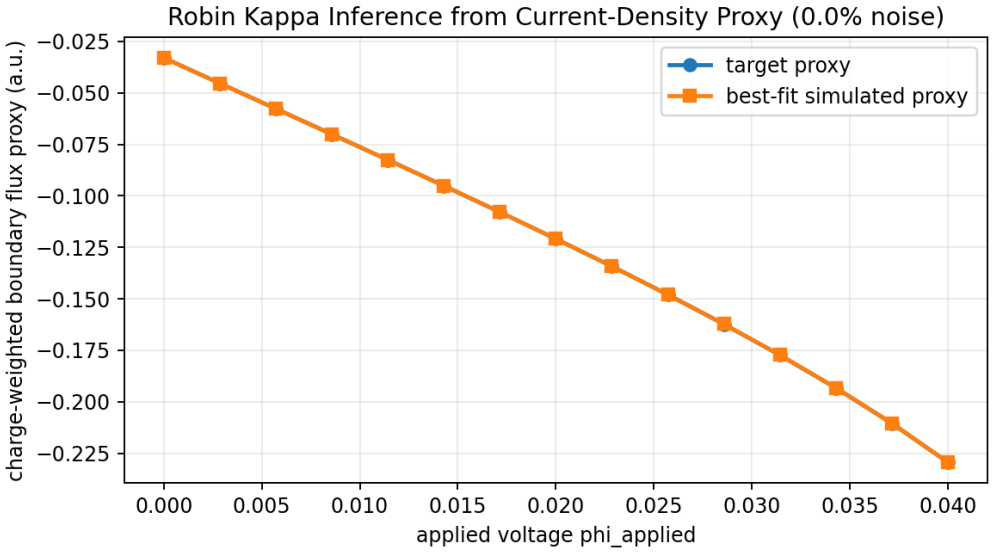

Figure 8: Robin κ current-density-proxy fit at 0% noise (arbitrary units).

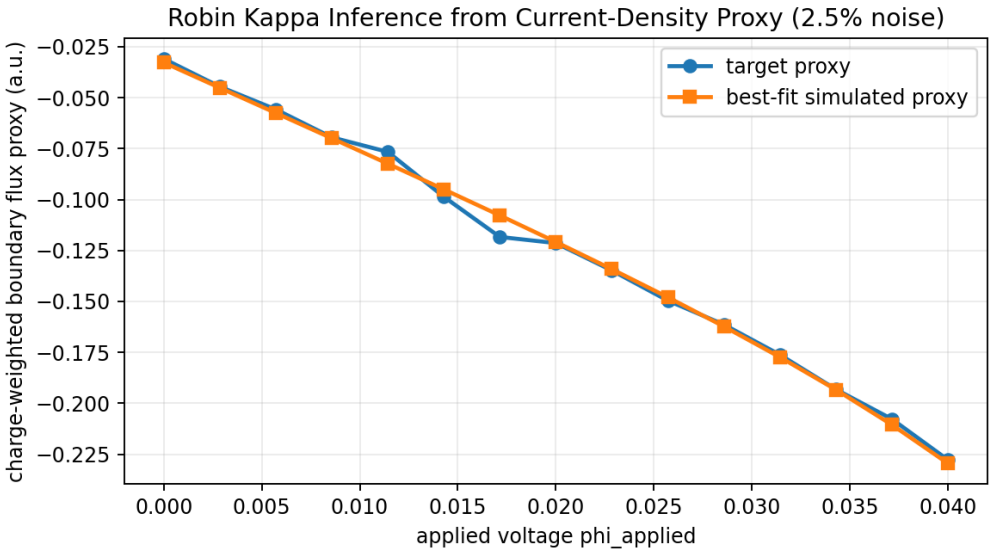

Figure 9: Robin κ current-density-proxy fit at 2.5% noise (arbitrary units).

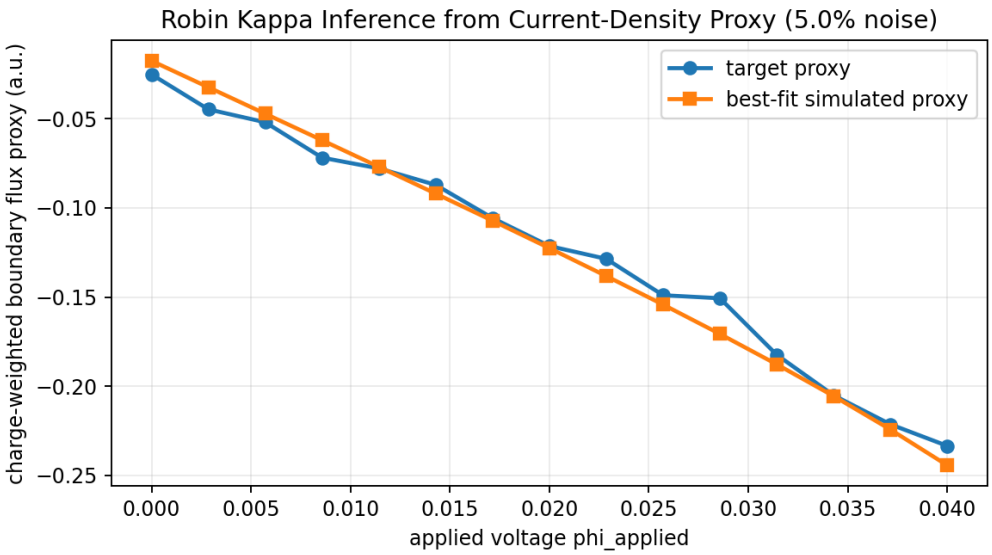

Figure 10: Robin κ current-density-proxy fit at 5% noise (arbitrary units).

7. Initial-Condition Runtime Optimization (Blob IC → Flat IC)

To reduce time-to-steady-state in each forward solve, a simple initial condition (uniform concentrations + linear potential) was benchmarked against the blob initial condition on the same no-noise Robin κ inference problem.

Case	Runtime (s)	Best κ_0	Best κ_1	Final Loss	SciPy Success
Blob IC	335.843	1.0015	1.9980	1.49×10^{-8}	True
Flat IC	310.872	1.0017	1.9980	1.39×10^{-8}	True

Table 2: No-noise runtime benchmark for Robin κ curve inference with identical optimizer settings.

Measured runtime improvement from this IC change:

$$335.843 \text{ s} \;\rightarrow\; 310.872 \text{ s},$$

which is a

$$1.0803\times \text{ speedup} \quad (7.44\% \text{ faster}).$$

Convergence quality remained effectively unchanged, so this is a low-risk performance optimization for data-generation and inference runs.

8. Replay Runtime/Memory Benchmark (Current-Density Proxy, No Noise)

What replay is doing (exactly)

The replay path exists to avoid rebuilding and re-solving all pointwise forward problems from scratch at every optimizer evaluation. Instead, for each $\phi_{\text{applied},j}$, we build a persistent taped model once and then re-evaluate that model under new κ values through `ReducedFunctional` replay.

Replay build phase (dynamic steady-state, not fixed-step)

For each sweep point j, replay bundle construction now does:

1. Start from an anchor κ (initially the optimizer initial guess, then rebuilt at reconverged κ when needed).
2. Run the full nonlinear forward solve with the same steady-state criterion used in non-replay solves.
3. Require the usual steady-state consecutive count plus an added buffer:

$$N_{\text{steady,target}} = N_{\text{steady,required}} + N_{\text{extra}},$$

   where $N_{\text{extra}} =$ `replay_extra_steady_steps` (default 3).
4. At the terminal replay-build state, tape four functionals per point:

$$\hat{O}_j(\kappa), \quad \hat{O}_{j,\text{prev}}(\kappa), \quad \hat{D}_j(\kappa) = \|U_j^{(N)} - U_j^{(N-1)}\|_{L^2}^2, \quad \hat{N}_j(\kappa) = \|U_j^{(N)}\|_{L^2}^2.$$

So replay is anchored at a state that already satisfied steady-state conditions, with extra post-steady timesteps to provide local robustness under nearby κ updates.

Per-evaluation replay diagnostics

At optimizer evaluation k, for each point j, replay computes:

$$O_j = \hat{O}_j(\kappa^{(k)}), \quad O_{j,\text{prev}} = \hat{O}_{j,\text{prev}}(\kappa^{(k)}), \quad \nabla_\kappa O_j,$$

and diagnostics

$$\Delta O_{j,\text{abs}} = |O_j - O_{j,\text{prev}}|,$$

$$\Delta O_{j,\text{rel}} = \frac{\Delta O_{j,\text{abs}}}{\max(|O_j|, |O_{j,\text{prev}}|, \varepsilon_{\text{abs}})},$$

$$\Delta U_{j,\mathrm{rel}} = \frac{\sqrt{\hat{D}_j(\kappa^{(k)})}}{\max(\sqrt{\hat{N}_j(\kappa^{(k)})}, 10^{-16})}.$$

A replay point is accepted as "steady-valid" only if:

- all replay outputs are finite/non-pathological, and
- $\Delta O_{j,\mathrm{rel}} \le \varepsilon_{\mathrm{rel}}$ or $\Delta O_{j,\mathrm{abs}} \le \varepsilon_{\mathrm{abs}}$.

If a point fails this check, it is marked non-converged for that evaluation. This is what prevents replay from silently drifting into non-steady behavior.

Failure handling and automatic rebuild

Replay handling is dynamic:

1. If replay throws an exception *or* any point fails replay diagnostics, replay is disabled for that candidate.
2. The same candidate κ is then evaluated via full resilient forward solves-to-steady-state (non-replay path).
3. After fallback reconverges (zero failed points), replay is rebuilt at the current effective κ and re-enabled (default: after 1 successful fallback evaluation).

This gives a fast path when replay is trustworthy, while preserving correctness by returning to full solves whenever replay looks invalid.

Auditability (new counters and artifact)

The run now records:

- `replay_rebuild_count`: total replay bundle builds/enables,
- `replay_diag_rebuild_count`: rebuilds triggered by failed replay diagnostics,
- `replay_exception_rebuild_count`: rebuilds triggered by replay exceptions.

These counts are printed in console summaries and written to:

`replay_diagnostics_summary.csv`.

Current status: replay comparison withdrawn

After adding dynamic replay diagnostics, replay-enabled runs still showed diagnostic failures at some optimizer evaluations (e.g., pointwise "replay not steady" failures) that triggered fallback-to-full-solve and automatic replay rebuild.
That behavior means a nominal "replay ON" run can become a hybrid trajectory mixing replay and non-replay evaluations. Because of that hybridity, the old ON/OFF runtime table is no longer a valid apples-to-apples comparison, and it has been removed from this report.
Therefore replay is now temporarily disabled in the active inference path until the replay validity issue is resolved. The replay diagnostics that motivated this decision remain implemented and logged (`replay_diagnostics_summary.csv`) for future debugging/re-enablement work.

9. Process-Parallel Point Solves

Implementation summary

To accelerate each objective/gradient evaluation across the 15 ϕ_{applied} points, point solves were parallelized with `ProcessPoolExecutor` (not threads). The key design choices were:

- **Process isolation for adjoint safety:** each worker is a separate process, so firedrake-adjoint tape state is not shared across concurrent point solves.
- **Spawn start method:** workers are launched with `spawn` to avoid inheriting unstable solver/tape state.
- **Static worker config:** baseline solver/steady settings are initialized once per worker, then each task supplies only (ϕ_{applied}, target, κ).
- **Serial fallback:** if worker-pool initialization or execution fails, the code falls back to serial point solves automatically.
- **User configurability:** request-level controls were added: `parallel_point_solves_enabled`, `parallel_point_workers`, `parallel_point_min_points`, and `parallel_start_method`.

Worker-count guess and benchmark

Hardware query on this machine reports 10 total cores split as 4 performance + 6 efficiency. For this adjoint/PDE-heavy workload, the working guess for optimal workers was 4 (match performance cores).
Benchmark protocol:

- Keep the same no-noise current-density-proxy inference setup (15 points, replay disabled, same target data and optimizer settings).
- Reuse the existing serial baseline from the prior benchmark data.

- Run one fresh parallel case with `parallel_point_workers` $= 4$.

Case	Runtime (s)	Peak RSS (MB)	Best κ_0	Best κ_1	Final Loss	Success
Serial baseline (existing)	279.642	423.9	0.9987	1.9981	1.0744×10^{-8}	True
Parallel (guess: 4 workers)	111.948	331.7	0.9987	1.9981	1.0744×10^{-8}	True

Table 3: No-noise current-density-proxy inference: existing serial baseline vs new process-parallel run (4 workers).

Observed impact:

$$\text{speedup (serial/parallel)} = 2.498\times$$

$$\text{memory ratio (parallel/serial)} = 0.782\times .$$

In this run, parallelization improved wall-clock time substantially while reaching essentially identical κ and loss.

10. Practical Takeaways from This Week

- The Robin flux-based inverse workflow is now fully implemented end-to-end and experimentally interpretable.
- A parallel current-density-proxy path is now in place: charge weighting is enforced, while absolute current scaling is explicitly deferred until unit calibration is finalized.
- The unified interface significantly reduced duplication and made target/solver swapping straightforward.
- Resilient retry mechanics now handle many nonlinear solve failures that previously aborted optimization.
- Regenerated proxy runs show strong recovery at 0% and 2.5% noise, with expected degradation and basin sensitivity at 5% noise.
- Replacing blob IC with a simple near-steady initial condition produced a measured 7.44% runtime reduction in no-noise Robin κ inference, with no meaningful loss of fit quality.
- Replay now has explicit steady-state diagnostics, fallback-to-full-solve, and rebuild counters, but is temporarily disabled in production inference runs because replay-enabled trajectories can still become invalid/hybrid and are not yet benchmark-comparable.
- Process-parallel point solves (with per-process tape isolation) produced a 2.498× runtime speedup against the existing serial baseline in the no-noise current-density-proxy benchmark, with matching fit quality.

- Reporting artifacts are now synchronized with the new branch: side-by-side data-generation plots, per-noise fit plots, summary table, and a 2.5% convergence GIF.

Future note: the next expansion target is a wider ϕ_{applied} sweep, including negative ϕ_{applied} values. Before expanding this range, runtime and solver-stability optimization is needed because a smaller time step Δt may be required to keep the forward solves stable in the more challenging regions.