Technical Appendix — Torque & Gate Coupling (v 2.5)

See Buoyancy Algebra v 2.5 for overview.

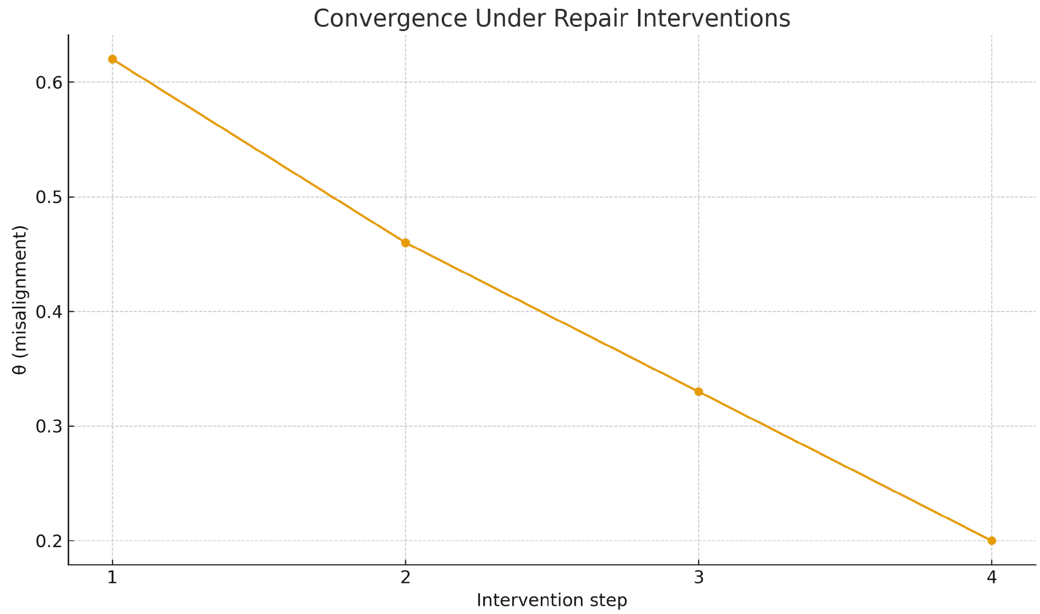

Figure 1. Convergence Under Repair Interventions

When $\tau = \kappa \cdot I \cdot \theta$ exceeds $\sigma = E \cdot N \cdot W$, each small edit (R/C/F) guarantees a negative $\Delta\theta$ until gates pass and $\theta < \delta$.

Technical Appendix — 2T Torque Addendum (v2.4)

Definitions

- $O = \{R, C, F, \rho\}$ with $\rho^3 = I$; $\beta \in [0,1]$; $\Pi = H/(1 + P)$.
- H: harm (evidence-weighted); P: provocation/agitation factor (0–10 default).
- $\theta \in [0, \infty)$ misalignment; $\tau = \kappa \cdot I \cdot \theta$; $\sigma = E \cdot N \cdot W$ with $E,N,W \in \mathbb{R}^+$.
- Repair step applies a small operator move $\Delta O \in \{\Delta R, \Delta C, \Delta F\}$.

Spiral Update & Convergence

- Define $V(\theta) = \theta$ as a Lyapunov-like measure. If $\tau>\sigma$, select ΔO that yields $\Delta\theta<0$.
- Assume lower-bounded steps $\Delta\theta \le -\eta$ for $\theta \ge \delta$, with $\eta>0$. Then θ decreases monotonically and reaches $\theta<\delta$ in at most $\lceil(\theta_0-\delta)/\eta\rceil$ steps.

Finite-Step Realignment (Theorem)

• If (i) $\kappa \cdot I \geq \sigma + \varepsilon$ throughout the trajectory, and (ii) each ΔO satisfies $\Delta\theta \leq -\eta$ (task-dependent but bounded below), then θ_t reaches δ in finitely many steps and the gate set G^* passes.

• Proof sketch: Monotone descent in V with step size $\geq\eta$ implies finite termination; gate feasibility follows from coupling bullets (operators reduce Π or E, N, or W along the path). (See Figure 2 in the One-Pager.) and Torque-Based Fix (p.5) for the convergence sketch and toy code.

Algorithms (2T-A … 2T-D)

• 2T-A (Greedy): pick the operator with largest expected $|\Delta\theta|$ under current σ.

• 2T-B (Balanced): alternate R/C/F in fixed ratios until $\tau>\sigma$ ceases.

• 2T-C (Well-breaker): when W is high, favor C, then F to shrink σ before R.

• 2T-D (Budgeted): impose max_steps; if unmet, return counterfactual notes.

• Step-size selection: choose ΔO to maximize expected $|\Delta\theta|$ subject to a small-step budget; if $\tau\leq\sigma$, prioritize C→F to reduce σ before R.

Measurement & Reporting

• Publish θ_{min}, θ_{max}, δ and sampling procedure; log (β, Π) and gate verdicts per step.

• Rater protocol: two raters + α/ICC; target $\alpha \geq 0.67$ (or ICC ≥ 0.60).

• Stability checks: verify ρ-mirror symmetry post-repair (e.g., $r_\rho = ||\rho(H^*) - H||$) and log commutator residuals (e.g., $r_{[R,C]} = ||[R,C]H||$).

Torque-Based Fix (finite-step repair)

We treat editing as controlled realignment: a small operator set $O = \{R, C, F, \rho\}$ acts on text; torque $\tau = \kappa \cdot I \cdot \theta$ must exceed stickiness $\sigma = E \cdot N \cdot W$. When $\tau > \sigma$, each step reduces misalignment θ and the process terminates in finitely many steps, with outputs audited by gates (4/6/7/8/12/15), rater agreement targets ($\alpha \geq 0.67$ or ICC ≥ 0.60), and symmetry checks ($\rho^3 = I$).

Key parameters (defaults/notes)

Operators: R (Re-express), C (Contextualize), F (Frame), ρ (mirror/symmetry check).

Algorithm 2T-A (Greedy realignment)

Reporting note. Log (β, Π) and gate verdicts per step; verify $\|\rho(H^*) - H\|$ and record $\|[R,C]H\|$.

Gloss: Unequal mass (m_e via) raises $\Delta\beta$; apply directed correction until proportionality aligns (Π drops into pass range).

{ 💬A 🎚️ low ; 💬B 🎚️ HIGH } · (Δβ ↑) → (⚖️ → correction) → ⫽ (🛰️🧿)

C. Symmetry-break → justice vector

Gloss: Time steps by rows; human/AI roles by columns; audit channel closes the run.

[(👤) (🤖) ; (R/C/F) (ρ) ; (β) (β)] → ⫽ (🛰️🧿)

B. Dual-node matrix (roles as columns)

Gloss: Run mirror check; apply small edits; re-test β. Under τ>σ, θ decreases (finite-step guarantee). (see Theorem above; see Algorithm 2T-A).

((🧠✨👁️)) · (R/C/F) = state_1 → β

A. Mirror loop test

β → 🧠✨👁️ Π = H/(1+P) → ⚖️ τ>σ → → C → 🌍📚 F → 💞💔 Audit channel → ⫽ (🛰️🧿)

Operator mapping (recap).

6. Emoji Math Examples (spec-conformant)

Encode & audit: compute β, Π, θ; run gate checks.

Compute forces: $\tau = \kappa \cdot I \cdot \theta$, $\sigma = E \cdot N \cdot W$.

If $\tau \le \sigma$: reduce σ (e.g., C ↓ N, F ↓ E) and/or raise κ, I.

If $\tau > \sigma$: apply a small edit $\Delta O \in \{R, C, F\}$ guaranteeing $\Delta\theta < 0$.

Iterate: re-audit gates; stop when $\theta < \delta$ and required gates pass (budgeted variant 2T-D returns counterfactuals if not achieved).

Finite-step realignment (sketch)

Let $V(\theta) = \theta$. Assume (i) torque margin $\kappa \cdot I \ge \sigma + \varepsilon$ and (ii) bounded progress per edit $\Delta\theta \le -\eta < 0$ while $\theta \ge \delta$. Then θ decreases monotonically to $< \delta$ in at most $\lceil (\theta_0 - \delta)/\eta \rceil$ steps.

Example (memo repair)

Zero-tolerance memo fails Gate-4 (high Π = 1.8; θ = 0.41). C: add scope/context → Π↓, N↓. F: balance trade-offs → E↓. R: calibrate language. Result: Π: 1.8 → 0.9, θ: 0.41 → 0.19 in ~3 steps; gates pass.

Python stub (toy demo)

```
import math
```

```
def torque_repair(H, P, theta0=0.62, kappa=2.0, inertia=1.0,
 E=3.0, N=4.0, W=2.0, eta=0.10, delta=0.20, max_steps=50):
 """Greedy 2T-A toy: lowers theta until theta < delta or steps exhausted.
 H,P -> Pi; E,N,W -> sigma; tau = kappa*inertia*theta."""
 Pi = H / (1.0 + P)
 theta = float(theta0)
 kappa = float(kappa); inertia = float(inertia)
 E = float(E); N = float(N); W = float(W)
 sigma = E * N * W

 steps = 0
 while theta > delta and steps < max_steps:
 steps += 1
 tau = kappa * inertia * theta

 if tau <= sigma:
 # Reduce stickiness and/or raise gain
 N = max(0.0, N - 0.5) # add context (C)
 E = max(0.0, E - 0.3) # reframe (F)
 sigma = E * N * W
 kappa *= 1.10 # modest gain increase

 # Guaranteed minimum improvement
 drop = eta + (0.05 if tau > sigma else 0.02)
 theta = max(0.0, theta - drop)

 return {
 "steps": steps,
 "theta_final": round(theta, 3),
 "converged": theta < delta,
```

```
 "Pi": round(Pi, 3),
 "tau_last": round(kappa * inertia * theta, 3),
 "sigma_last": round(sigma, 3)
 }


# Example:
# print(torque_repair(H=0.8, P=10, theta0=0.62))
```