OPERATION KOMOREBI

REALIZATION CALCULUS



RESEARCH LIBRARY

Certified Bayesian Inference

One-observation Gaussian updates, exact packets, and outward-rounded evidence

Lean and evidence guide

Komorebi P.

Standardized publication edition / 8 October 2026

REALIZATION CALCULUS

Reading scope

An executable proof map for the certified one-observation Gaussian solver. Actual definitions and complete proof excerpts are paired with their hypotheses, source locations, and evidence boundaries.

The reading contract

This companion explains how Certified Inference Core 1.0.0 turns finite rational model data into a returned gain, a posterior transport guarantee, and outward-rounded parameter intervals. It follows the actual project sources rather than presenting a new implementation. Every source listing is copied verbatim from the archived release, with its file and inclusive line span. Complete proof listings retain the entire declaration and proof body. Ambient namespaces, imports, finite-type instances, and previously declared variables are supplied by their source module and are not repeated in every listing. The listings become executable by opening their original modules in the pinned project.

The central entry point is `Komorebi.CachedPosterior.runCached`. Its selected gain updates and residual cache are exact rational operations. Its final certificate and output arrays are evaluated with outward-rounded dyadic arithmetic. A separate retained branch performs rounded full-gain updates. These are two different algorithms with different convergence arguments.

The archive records 117 successful active-module compilations and axiom reports for 543 named theorem or lemma declarations. Those are archived kernel checks from 12 September 2026. Fresh publication checks re-read the source bytes, manifest, successful receipts, attempted-source snapshots, logs, declaration locations, import records, and the listings printed here. A pinned Lean compiler was unavailable during this publication session, so no fresh Lean compilation is claimed. The archive contains no compiled `.olean` payloads. Their output hashes are preserved in receipts and agree across recorded dependency links, but cannot be rehashed from absent binaries.

The scope is finite-dimensional inference for one observation update. Model inputs are exact rational factors, the noise floor is positive, packet geometry is fixed, and posterior/output tolerances are positive rationals. Singular priors and rectangular observation operators are allowed. Returned intervals enclose the parameters of the certified Joseph Gaussian. They do not separately certify the Gaussian obtained by replacing those parameters with interval centers.

Contents

1 Start with the returned object **3**

2 Soundness and guaranteed success are distinct theorems **4**

3 The exact model is represented by factors **5**

4 Why the target is a Bayesian posterior **6**

5 A residual controls distribution error **7**

6 Packet geometry is checked, including completeness **8**

7 The finite checker and its semantic theorem 9

8 Selected updates preserve unselected packets 10

9 Maximum selection is an executed finite scan 11

10 Finite fuel reserves verification slack 12

11 The cache invariant is proved for the executed update 13

12 Cached execution equals the recomputed greedy trajectory 14

13 Search completeness closes the algorithmic existence step 15

14 Outward dyadic arithmetic has its own soundness proof 17

15 Finite verification precision is computed, not guessed 18

16 Output radius and posterior tolerance mean different things 19

17 Execution example I: a rectangular coordinate model 20

18 Execution example II: a singular prior and rotated packets 21

19 A checked counterexample prevents a false rounded-cache claim 23

20 Negative controls test specific failure boundaries 23

21 The rounded full-gain branch remains a separate result 24

22 Rebuild the pinned project without overwriting the archive 25

23 How to extend the proof without weakening the contract 26

A Module order and declaration map 27

B Evidence that accompanies the source 30

1 Start with the returned object

A proof-oriented interface must say what a successful return means before asking how the program reaches one. The output record stores a verification precision, a rational candidate, a separate parameter precision, and interval arrays. The candidate contains the gain and the local energy bounds used in the distribution certificate. Precision fields identify the arithmetic graph that produced the certificate and arrays.

Komorebi/PosteriorOutputSolver.lean, lines 10–14.

```
structure Result (ι κ α β : Type*) where
  verificationPrecision : ℕ
  candidate : RationalCandidate ι κ α β
  parameterPrecision : ℕ
  arrays : Arrays ι
```

Komorebi/PosteriorOutputSolver.lean, lines 28–36.

```
noncomputable def ResultValid (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (ε δ : ℚ) (r : Result ι κ α β) : Prop :=
  checkCandidate M Q D r.candidate = true ∧
  checkTolerance M r.candidate ε = true ∧
  wasserstein2 M.posterior (M.approximation r.candidate.gain) ≤ ENNReal.ofReal (ε : ℝ) ∧
  EnclosesParameters M r.candidate.gain r.arrays ∧ RadiiAtMost r.arrays δ ∧
  r.parameterPrecision = outputPrecision M r.candidate.gain δ ∧
  r.arrays = evaluate r.parameterPrecision M r.candidate.gain
```

`ResultValid` has seven conjuncts. The first two assert that the finite rational candidate and tolerance decisions pass. The third states the real distribution guarantee. The fourth and fifth assert coordinate enclosure and a uniform radius cap for the approximate Gaussian's parameters. The last two identify the selected output precision and actual evaluated arrays. A consumer can project the obligation it needs without interpreting a printed diagnostic string as evidence.

The transport distance in source is extended nonnegative real valued. Thus `ENNReal.ofReal` embeds the rational tolerance into that codomain. This does not change the practical inequality $W_2 \le \varepsilon$ under the established nonnegativity hypotheses. It does avoid treating a potentially infinite distance as an ordinary real number before finiteness has been proved.

Komorebi/CachedPacketOutput.lean, lines 11–18.

```
def runCached (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (seed : α) (scan : List α) (ε δ : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ) : Option (Result ι κ α β)
     :=
  if 0 < δ then
    match solveCached M Q D seed scan ε probes B with
    | none => none
    | some (p,c) => some ⟨p,c,outputPrecision M c.gain δ,represent M c.gain δ⟩
  else none
```

The executable function first rejects a nonpositive output tolerance, then calls the cached solver. A successful candidate receives a computed parameter precision and represented arrays. A successful return is a logical event in this exact Lean program. The separate Python producer uses a provisional status until an emitted explicit instance is compiled.

2 Soundness and guaranteed success are distinct theorems

The successful-return theorem does not ask callers to supply packet or noise validity as separate hypotheses. Those obligations are enforced by the executed checks and construction. Its only substantive input premise is an equality showing that this invocation returned this result.

Komorebi/CachedPacketOutput.lean, lines 34–51.

```
theorem runCached_sound (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (seed : α) (scan : List α) (ε δ : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ) (r : Result ι κ α β)
    (hr : runCached M Q D seed scan ε δ probes B = some r) : ResultValid M Q D ε δ r := by
  unfold runCached at hr
  split at hr
  next hδ =>
    cases hs : solveCached M Q D seed scan ε probes B with
    | none => simp only [hs] at hr; contradiction
    | some pc =>
      obtain ⟨p,c⟩ := pc
      simp only [hs, Option.some.injEq] at hr
      subst r
      obtain ⟨hc,ht⟩ := solveCached_sound M Q D seed scan ε probes B p c hs
      obtain ⟨he,hprec⟩ := represent_spec M c.gain δ hδ
      exact ⟨hc,ht,cached_solver_return_bounds_posterior M Q D seed scan ε probes B p c hs,
        he,hprec,rfl,rfl⟩
  next => contradiction
```

The proof unfolds the program, separates the rejected branch, analyzes the optional cached-solver return, and combines three established facts: the solver's candidate/tolerance soundness, its posterior bound, and the interval representation specification. The use of `rfl` in the last conjuncts verifies origin, not numerical approximation.

Guaranteed success needs the mathematically admissible input class. It assumes a positive noise floor, valid complete output and input packet families, a supplied output seed, a scan covering every output label, and positive posterior and output tolerances. The initial gain and finite list of requested verification precisions are arbitrary. An appended sufficient-precision fallback removes dependence on the caller guessing a successful precision.

Komorebi/CachedPacketOutput.lean, lines 20–31.

```
theorem runCached_terminates (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (hQ : RationalPacketsValid Q) (hD : RationalPacketsValid D)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan)
    (ε δ : ℚ) (hε : 0 < ε) (hδ : 0 < δ) (probes : List ℕ) (B : Matrix ι κ ℚ) :
    ∃ r, runCached M Q D seed scan ε δ probes B = some r ∧ ResultValid M Q D ε δ r := by
  obtain ⟨p,c,hret,hc,ht,hw⟩ :=
    cached_solver_reaches_posterior_tolerance M hs Q D hQ hD seed scan hscan ε hε probes B
  refine ⟨⟨p,c,outputPrecision M c.gain δ,represent M c.gain δ⟩, ?_, ?_⟩
  · simp only [runCached, ite_eq_left hδ, hret]
  · obtain ⟨he,hr⟩ := represent_spec M c.gain δ hδ
    exact ⟨hc,ht,hw,he,hr,rfl,rfl⟩
```

This theorem proves that the program returns *some* valid result. It therefore closes an existence obligation for the algorithm's own output, rather than saying only that outputs would be safe if produced. The finite fuel can be conservative. Mathematical success on unbounded exact integers/rationals does not

promise that the bounded reference producer completes within a machine's time or memory limit.

Komorebi/CachedPacketOutput.lean, lines 53–57.

```
theorem runCached_rejects_nonpositive_output_tolerance (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (seed : α) (scan : List α) (ε δ : ℚ) (hδ : δ ≤ 0) (probes : List ℕ) (B : Matrix ι κ ℚ) :
    runCached M Q D seed scan ε δ probes B = none := by
  simp only [runCached, ite_eq_right (not_lt.mpr hδ)]
```

3 The exact model is represented by factors

The four finite types separate state coordinates ι, observation coordinates κ, prior-factor columns υ, and noise-factor columns ω. Their cardinalities need not coincide. Factors make covariance positivity a theorem about construction. They do not put an unchecked positive-definiteness flag inside the input record.

Komorebi/RationalCertificateChecker.lean, lines 8–14.

```
structure RationalModel (ι κ υ ω : Type*) where
  observationMatrix : Matrix κ ι ℚ
  priorFactor : Matrix ι υ ℚ
  noiseFactor : Matrix κ ω ℚ
  noiseFloor : ℚ
  mean : ι → ℚ
  observation : κ → ℚ
```

Komorebi/RationalCertificateChecker.lean, lines 28–29.

```
def prior (M : RationalModel ι κ υ ω) : Matrix ι ι ℚ :=
  M.priorFactor * M.priorFactor.transpose
```

Komorebi/RationalCertificateChecker.lean, lines 31–32.

```
def noise (M : RationalModel ι κ υ ω) : Matrix κ κ ℚ :=
  M.noiseFactor * M.noiseFactor.transpose + M.noiseFloor • 1
```

Komorebi/RationalCertificateChecker.lean, lines 34–35.

```
def innovation (M : RationalModel ι κ υ ω) : Matrix κ κ ℚ :=
  M.observationMatrix * M.prior * M.observationMatrix.transpose + M.noise
```

Komorebi/RationalCertificateChecker.lean, lines 37–38.

```
def residual (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) : Matrix ι κ ℚ :=
  M.prior * M.observationMatrix.transpose - B * M.innovation
```

Write $P = LL^{\mathsf{T}}$, $R = TT^{\mathsf{T}} + \sigma I$, $S = HPH^{\mathsf{T}} + R$, and $F(B) = PH^{\mathsf{T}} - BS$. Positive σ makes R positive definite even if T has deficient rank. The prior P may remain singular. The exact posterior gain involves S^{-1}, but iteration and stopping can evaluate $F(B)$ without constructing that inverse.

Rational entries embed into actual finite-dimensional Euclidean spaces and continuous linear maps. The source proves compatibility of casts with matrix operations. The meaning of the rational model is consequently a real probability measure, not a symbolic tuple whose probabilistic interpretation is assumed later.

Komorebi/RationalModelSemantics.lean, lines 69–71.

```
def posterior (M : RationalModel ι κ υ ω) : Measure (EuclideanSpace ℝ ι) :=
  fdPosteriorKernel (realMatrix M.observationMatrix) (realVector M.mean)
    (realMatrix M.prior) (realMatrix M.noise) (realVector M.observation)
```

Komorebi/RationalModelSemantics.lean, lines 74–76.

```
def approximation (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) : Measure (EuclideanSpace ℝ ι) :=
  gainPosteriorLaw (realMatrix M.observationMatrix) (realVector M.mean)
    (realMatrix M.prior) (realMatrix M.noise) (realMatrix B) (realVector M.observation)
```

The target is the explicitly selected Gaussian posterior kernel evaluated at the supplied observation. The approximation is the gain-dependent Joseph law. All distribution statements below refer to those concrete measures.

4 Why the target is a Bayesian posterior

The Gaussian semantics layer builds both orders of the same joint law. In the forward description, a prior draw passes through the affine Gaussian likelihood. In the reverse description, an observation draw passes through the displayed posterior kernel. Equality of those actual joint measures supplies disintegration.

Komorebi/FDGaussianConditioning.lean, lines 11–21.

```
theorem fd_joint_disintegration
    (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ)
    (hP : P.PosSemidef) (hR : R.PosDef) :
    (multivariateGaussian (matrixCLM H m) (innovationMatrix H P R)) ⊗ₘ
        fdPosteriorKernel H m P R =
      ((multivariateGaussian m P) ⊗ₘ gaussianAffineKernel H 0 R).map Prod.swap := by
  apply map_pack_injective
  rw [reverse_joint_gaussian H m P R hP hR,
    forward_joint_gaussian H m P R hP hR.posSemidef,
    reverseJointCov_eq_forwardJointCov H P R hP hR]
```

Taking the observation marginal identifies the evidence law. Mathlib's canonical posterior is a version of conditional probability, so its identification is an almost-everywhere statement under that evidence law. A conditional probability version can be modified on an evidence-null set without changing disintegration. The source retains this distinction rather than changing the quantifier to pointwise equality.

Komorebi/FDGaussianConditioning.lean, lines 34–46.

```
theorem fdPosteriorKernel_eq_canonical
    (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ)
    (hP : P.PosSemidef) (hR : R.PosDef) :
    fdPosteriorKernel H m P R =ᵐ[
      multivariateGaussian (matrixCLM H m) (innovationMatrix H P R)]
      posterior (gaussianAffineKernel H 0 R) (multivariateGaussian m P) := by
  have h := ae_eq_posterior_of_compProd_eq
    (κ := gaussianAffineKernel H 0 R) (μ := multivariateGaussian m P)
    (η := fdPosteriorKernel H m P R) (by
      rw [fd_observation_evidence H m P R hP hR]
      exact fd_joint_disintegration H m P R hP hR)
  rwa [fd_observation_evidence H m P R hP hR] at h
```

The solver certificate is pointwise for the chosen explicit kernel at the supplied rational observation. The mathematical statement relating this kernel to a canonical posterior version is evidence-almost everywhere. These are compatible conclusions. They answer different version questions.

Singular priors require no prior inverse in this chain. The positive-definite noise supplies the invertible innovation covariance. This is why the rotated example later can have rank-one prior support while its observation covariance is nonsingular.

Komorebi/FDGaussianConditioning.lean, lines 63–73.

```
theorem canonicalPosterior_eq_josephGaussian
    (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ)
    (hP : P.PosSemidef) (hR : R.PosDef) :
    ∀ᵐ y ∂multivariateGaussian (matrixCLM H m) (innovationMatrix H P R),
      posterior (gaussianAffineKernel H 0 R) (multivariateGaussian m P) y =
        multivariateGaussian
          (m + matrixCLM (gainMatrix H P R) (y - matrixCLM H m))
          (josephCovMatrix H P R) := by
  rw [josephCovMatrix_eq_posteriorCovMatrix H P R hP hR]
  exact canonicalPosterior_eq_multivariateGaussian H m P R hP hR
```

5 A residual controls distribution error

The proof couples the exact posterior and approximate Joseph law using shared Gaussian ingredients. Its squared transport cost is derived from that actual coupling. Writing $K = PH^{\mathsf{T}}S^{-1}$ and $v = y - Hm$, the coupling cost is

$$\|(K - B)v\|^2 + \operatorname{tr}((K - B)S(K - B)^{\mathsf{T}}).$$

The identity $K - B = F(B)S^{-1}$ converts this to a residual expression. This is an upper bound on the optimal transport cost, since a particular coupling need not minimize the transport problem.

Komorebi/FDInexactGainCoupling.lean, lines 24–35.

```
theorem inexactGainCoupling_cost_residual (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ) (B : Matrix ι κ ℝ)
    (y : EuclideanSpace ℝ κ) (hP : P.PosSemidef) (hR : R.PosDef) :
    quadraticCost (inexactGainCoupling H m P R B y) =
      ENNReal.ofReal
        (‖matrixCLM (gainResidual H P R B * (innovationMatrix H P R)⁻¹)
            (y-matrixCLM H m)‖^2 +
          (gainResidual H P R B * (innovationMatrix H P R)⁻¹ *
            (gainResidual H P R B).transpose).trace) := by
  rw [inexactGainCoupling_cost_gap H m P R B y hP hR,
    ← residual_cov_eq_gap_cov H P R B hP hR,
    gain_gap_eq_residual_mul_inv H P R B hP hR]
```

The noise floor gives $\|S^{-1}\| \le \sigma^{-1}$. A Frobenius/operator-norm majorant then removes the inverse from the final stopping quantity:

$$W_2^2(\pi_y, \widehat{\pi}_{B,y}) \le \|F(B)\|_{\mathrm{F}}^2 \frac{\sigma + \|v\|^2}{\sigma^2}.$$

The right side is computed from the current gain, model factors, and observation. The source's explicit hypotheses make the roles of covariance positivity, positive noise floor, and its quadratic lower bound visible.

Komorebi/FDResidualEnergyCertificate.lean, lines 39–49.

```
theorem posterior_wasserstein2Sq_le_residual_energy
    (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ) (B : Matrix ι κ ℝ)
    (y : EuclideanSpace ℝ κ) (hP : P.PosSemidef) (hR : R.PosDef)
    (σ : ℝ) (hσ : 0 < σ) (hfloor : QuadraticLowerBound R σ) :
    wasserstein2Sq (fdPosteriorKernel H m P R y) (gainPosteriorLaw H m P R B y) ≤
      ENNReal.ofReal (gainResidualEnergyBudget H m P R B y σ) := by
  apply (posterior_wasserstein2Sq_le_gain_residual H m P R B y hP hR).trans
  apply ENNReal.ofReal_le_ofReal
  exact quadratic_residual_cost_le_energy _ _ _ (σ⁻¹)
    (innovation_inverse_norm_le H P R hP hR σ hσ hfloor)
```

No comparison to only a formal mean/covariance surrogate is used. The left side contains the actual target and approximation measures. A zero residual yields equality with the selected posterior law, while a small residual yields a quantitative whole-law certificate.

Komorebi/FDInexactGainCoupling.lean, lines 54–64.

```
theorem gainPosteriorLaw_eq_of_residual_zero
    (H : Matrix κ ι ℝ) (m : EuclideanSpace ℝ ι)
    (P : Matrix ι ι ℝ) (R : Matrix κ κ ℝ) (B : Matrix ι κ ℝ)
    (y : EuclideanSpace ℝ κ) (hP : P.PosSemidef) (hR : R.PosDef)
    (hF : gainResidual H P R B = 0) :
    gainPosteriorLaw H m P R B y = fdPosteriorKernel H m P R y := by
  have hg := gain_gap_eq_residual_mul_inv H P R B hP hR
  rw [hF, Matrix.zero_mul] at hg
  have heq : gainMatrix H P R = B := sub_eq_zero.mp hg
  rw [← heq]
  exact gainPosteriorLaw_exact H m P R y hP hR
```

6 Packet geometry is checked, including completeness

Output packets Q_a partition state directions. Input packets D_b partition observation directions for local residual certification. Each family is a finite rational self-adjoint orthogonal resolution of the identity. Four obligations are checked: symmetry, idempotence, pairwise annihilation, and completeness. Missing the completeness clause could discard residual energy outside the listed packets.

Komorebi/RationalPacketCertificates.lean, lines 13–17.

```
abbrev RationalPacketsValid (Q : α → Matrix ι ι ℚ) : Prop :=
  (∀ a, (Q a).transpose = Q a) ∧
  (∀ a, Q a * Q a = Q a) ∧
  (∀ a b, a ≠ b → Q a * Q b = 0) ∧
  (∑ a, Q a) = 1
```

Komorebi/RationalPacketCertificates.lean, lines 19–20.

```
def checkRationalPackets (Q : α → Matrix ι ι ℚ) : Bool :=
  decide (RationalPacketsValid Q)
```

Komorebi/RationalPacketCertificates.lean, lines 27–32.

```
noncomputable def realPackets (Q : α → Matrix ι ι ℚ) (h : RationalPacketsValid Q) :
    SelfAdjointPackets ι α where
  proj a := realMatrix (Q a)
  symmetric a := by rw [← realMatrix_transpose, h.1 a]
```

```
  idempotent a := by rw [← realMatrix_mul, h.2.1 a]
  disjoint a b hab := by rw [← realMatrix_mul, h.2.2.1 a b hab, realMatrix_zero]
```

The rational predicates construct real projector geometry using exact casts. In Euclidean space, self-adjoint disjoint projections give an energy partition. In particular, the sum of output-packet squared Frobenius energies equals the entire residual energy. The two-sided packet theorem similarly accounts for both output and observation directions in the final certificate.

Komorebi/PacketSelectiveAlgebra.lean, lines 13–21.

```
theorem sum_left_energy (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (F : Matrix ι κ ℚ) : (∑ a, rationalEnergy (Q a * F)) = rationalEnergy F := by
  have h := matrixEnergy_sumOn_left (realPackets Q hQ) Finset.univ (realMatrix F)
  have hc := realPackets_complete Q hQ
  unfold SelfAdjointPackets.Complete at hc
  rw [hc, Matrix.one_mul] at h
  have hh : (rationalEnergy F : ℝ) = ∑ a, (rationalEnergy (Q a * F) : ℝ) := by
    simpa only [realPackets_proj, ← realMatrix_mul, ← rationalEnergy_cast] using h
  exact_mod_cast hh.symm
```

Packet labels can include zero projectors. What matters is the proved resolution. The seed supplies an actual label, so the chosen-output family is nonempty for the selection/convergence theorem. Duplicate scan entries are allowed. Scan coverage concerns which labels can be found, not a claim that every scanned label has positive energy.

7 The finite checker and its semantic theorem

An arbitrary producer may propose a gain and conservative local upper bounds. The finite candidate predicate requires positivity, both complete packet families, every local residual-energy bound, and an innovation-vector energy bound. The checker decides those exact rational obligations.

Komorebi/RationalCertificateChecker.lean, lines 17–20.

```
structure RationalCandidate (ι κ α β : Type*) where
  gain : Matrix ι κ ℚ
  packetUpper : α → β → ℚ
  innovationUpper : ℚ
```

Komorebi/RationalCertificateChecker.lean, lines 51–56.

```
abbrev ValidCandidate (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (c : RationalCandidate ι κ α β) : Prop :=
  0 < M.noiseFloor ∧ RationalPacketsValid Q ∧ RationalPacketsValid D ∧
  (∀ a b, rationalEnergy (Q a * M.residual c.gain * D b) ≤ c.packetUpper a b) ∧
  rationalVectorEnergy M.innovationVector ≤ c.innovationUpper
```

Komorebi/RationalCertificateChecker.lean, lines 59–61.

```
def checkCandidate (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (c : RationalCandidate ι κ α β) : Bool := decide (ValidCandidate M Q D c)
```

Komorebi/RationalCertificateChecker.lean, lines 70–72.

```
def checkTolerance (M : RationalModel ι κ υ ω)
    (c : RationalCandidate ι κ α β) (ε : ℚ) : Bool :=
  decide (0 ≤ ε ∧ totalUpper c * (M.noiseFloor+c.innovationUpper) ≤ ε^2*M.noiseFloor^2)
```

The tolerance test avoids a square root. With $U = \sum_{a,b} u_{ab}$ and V the innovation upper bound, it checks $U(\sigma + V) \le \varepsilon^2\sigma^2$ and nonnegative ε. The semantic theorem links passing Boolean decisions to the target law. Positivity of denominators is recovered from candidate validity, so algebraic division never silently assumes a nonzero noise floor.

Komorebi/RationalPosteriorCertificate.lean, lines 55–65.

```
theorem checked_tolerance_bounds_posterior (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ) (c : RationalCandidate ι κ α β)
    (ε : ℚ) (hc : checkCandidate M Q D c = true) (ht : checkTolerance M c ε = true) :
    wasserstein2 M.posterior (M.approximation c.gain) ≤ ENNReal.ofReal (ε : ℝ) := by
  have h := (checkCandidate_eq_true M Q D c).mp hc
  have ht' := (checkTolerance_iff_score M c ε h.1).mp ht
  apply wasserstein2_le_of_sq_le _ _ (ε : ℝ) (by exact_mod_cast ht'.1)
  apply (checked_score_bounds_posterior M Q D c hc).trans
  apply ENNReal.ofReal_le_ofReal
  have hh : (candidateScore M c : ℝ) ≤ ((ε^2 : ℚ) : ℝ) := by exact_mod_cast ht'.2
  simpa [pow_two] using hh
```

The final rounded verifier does not execute the full exact-residual `checkCandidate`. It checks setup, constructs outward bounds, and executes the tolerance test. Construction soundness proves that its candidate would pass the exact checker. This operational distinction matters when counting full residual recomputations and describing the arithmetic boundary.

8 Selected updates preserve unselected packets

Let $\eta = (\sigma + \operatorname{tr} S)^{-1}$ be the exact rational step. A selected update changes B by $\eta Q_a F(B)$. Its residual satisfies

$$F(B^+) = F(B) - \eta Q_a F(B) S.$$

Multiplying on the left by Q_b kills the change whenever $b \neq a$, because $Q_b Q_a = 0$. For the selected packet, idempotence gives the right innovation transition $I - \eta S$. These two identities justify the cache.

Komorebi/PacketSelectiveAlgebra.lean, lines 25–26.

```
def packetStep (M : RationalModel ι κ υ ω) (Q : Matrix ι ι ℚ)
    (B : Matrix ι κ ℚ) : Matrix ι κ ℚ := B + M.solverStep • (Q * M.residual B)
```

Komorebi/PacketSelectiveAlgebra.lean, lines 37–43.

```
theorem unselected_residual (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (a b : α) (hba : b ≠ a) (B : Matrix ι κ ℚ) :
    Q b * M.residual (packetStep M (Q a) B) = Q b * M.residual B := by
  rw [packetStep_residual]
  simp only [Matrix.mul_sub, Matrix.mul_smul, ← Matrix.mul_assoc,
    hQ.2.2.1 b a hba, Matrix.zero_mul, smul_zero, sub_zero]
```

Komorebi/PacketSelectiveAlgebra.lean, lines 47–53.

```
theorem selected_residual (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (a : α) (B : Matrix ι κ ℚ) :
    Q a * M.residual (packetStep M (Q a) B) =
      (Q a * M.residual B) * (1 - M.solverStep • M.innovation) := by
  rw [packetStep_residual]
  simp only [Matrix.mul_sub, Matrix.mul_one, Matrix.mul_smul, ← Matrix.mul_assoc, hQ.2.1 a]
```

Only output locality is asserted. An input subpacket can couple to other input directions through the innovation matrix. The retained `input_packet_effect` identity contains SD_b, and does not replace it with a diagonal or commuting expression. General input-block locality would need additional hypotheses.

The spectral transition contracts the selected energy by the proved full-update rate. Summing the energy partition produces global descent weighted by the energy of the chosen packet. This is the bridge from local algebra to a terminating selection policy.

9 Maximum selection is an executed finite scan

The algorithm selects by exact rational energy. It does not assume a magical maximum oracle or a predetermined cyclic schedule. The recursive scan compares the incumbent energy to the next label and replaces it only under a strict inequality. Ties retain the incumbent.

Komorebi/GreedyPacketSelection.lean, lines 9–11.

```
def winner (e : α → ℚ) : α → List α → α
  | a, [] => a
  | a, b::rest => winner e (if e a < e b then b else a) rest
```

Komorebi/GreedyPacketSelection.lean, lines 48–51.

```
theorem selectPacket_max (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan) (B : Matrix ι κ ℚ) (a : α) :
    rationalEnergy (Q a*M.residual B) ≤ rationalEnergy (Q (selectPacket M Q seed scan B)*M.residual B)
    :=
  winner_ge_member (fun b => rationalEnergy (Q b*M.residual B)) seed a scan (hscan a)
```

Komorebi/GreedyPacketSelection.lean, lines 55–65.

```
theorem selected_energy_share (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan) (B : Matrix ι κ ℚ) :
    rationalEnergy (M.residual B) / (Fintype.card α : ℚ) ≤
      rationalEnergy (Q (selectPacket M Q seed scan B)*M.residual B) := by
  have hc : (0 : ℚ) < Fintype.card α := by
    exact_mod_cast (Fintype.card_pos_iff.mpr ⟨seed⟩)
  have hsum := Finset.sum_le_sum (fun a (_ : a ∈ Finset.univ) => selectPacket_max M Q seed scan hscan B
    a)
  rw [sum_left_energy Q hQ] at hsum
  simp only [Finset.sum_const, Finset.card_univ, nsmul_eq_mul] at hsum
  exact (div_le_iff₀ hc).mpr (by simpa only [mul_comm] using hsum)
```

Coverage of the scan implies that the executed winner has at least every packet's energy. If there are A output labels, it therefore captures at least $1/A$ of the total energy. Repeated labels do not change this

pigeonhole argument. The supplied seed ensures $A > 0$.

Combining this share with selected-packet descent yields the adaptive energy factor

$$q_{\text{adapt}} = 1 - \frac{1-q}{A}, \qquad q = \left(1 - \frac{\sigma}{\sigma + \operatorname{tr} S}\right)^2.$$

It lies in $[0, 1)$ under the admissible model hypotheses. This is a conservative rate bound. It need not equal the measured contraction of a particular packet or trajectory.

Komorebi/GreedyPacketSelection.lean, lines 69–69.

```
def adaptiveRate (q : ℚ) (count : ℕ) : ℚ := 1 - (1-q)/(count : ℚ)
```

Komorebi/GreedyPacketConvergence.lean, lines 40–48.

```
theorem greedy_score_geometric (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan) (B : Matrix ι κ ℚ) (n : ℕ) :
    M.residualScore (greedyGain M Q seed scan B n) ≤
      (adaptiveRate M.solverRate (Fintype.card α))^n*M.residualScore B := by
  have hf : 0 ≤ (M.noiseFloor+rationalVectorEnergy M.innovationVector)/M.noiseFloor^2 :=
    div_nonneg (add_nonneg hs.le (rationalVectorEnergy_nonneg _)) (sq_nonneg _)
  have h := mul_le_mul_of_nonneg_right (greedy_energy_geometric M hs Q hQ seed scan hscan B n) hf
  simpa only [RationalModel.residualScore, div_eq_mul_inv, mul_assoc] using h
```

10 Finite fuel reserves verification slack

The solver uses an explicit rational fuel expression, avoiding logarithm evaluation and an eigenvalue oracle. A Bernoulli-type envelope turns a geometric bound into a simple ceiling. The resulting fuel can substantially overestimate the update count that early stopping actually uses.

Komorebi/GeometricFuel.lean, lines 28–28.

```
def toleranceFuel (q a ε : ℚ) : ℕ := Nat.ceil (a / (ε^2 * (1-q)))
```

Komorebi/GreedyPacketConvergence.lean, lines 51–52.

```
def adaptiveFuel (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) (ε : ℚ) (count : ℕ) : ℕ :=
  toleranceFuel (adaptiveRate M.solverRate count) (M.residualScore B) (ε/2)
```

Komorebi/GreedyPacketConvergence.lean, lines 54–65.

```
theorem greedy_score_at_fuel (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (hQ : RationalPacketsValid Q)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan) (B : Matrix ι κ ℚ)
    (ε : ℚ) (hε : 0 < ε) (n : ℕ) (hn : adaptiveFuel M B ε (Fintype.card α) ≤ n) :
    M.residualScore (greedyGain M Q seed scan B n) ≤ ε^2/4 := by
  have hr := adaptiveRate_bounds M.solverRate (Fintype.card α)
    (M.solverRate_bounds hs).1 (M.solverRate_bounds hs).2 (Fintype.card_pos_iff.mpr ⟨seed⟩)
  have h := (greedy_score_geometric M hs Q hQ seed scan hscan B n).trans
    (geometric_fuel_sufficient _ _ (ε/2) hr.1 hr.2 (by positivity) n hn)
  calc
    _ ≤ (ε/2)^2 := h
    _ = ε^2/4 := by ring
```

The cached score aims at $\varepsilon^2/4$, not merely ε^2. Once the exact score crosses that stricter threshold, the sufficient final precision can tolerate up to another $\varepsilon^2/4$ of score inflation and still pass the desired target.

The construction therefore has ample strict slack. It does not depend on the rounded score decreasing monotonically with every precision increment.

The theorem permits every iterate at or above the computed fuel. The operational search stops at the first passing exact score, which may be much earlier. The coordinate example uses six updates despite a fuel of 410,093, and the rotated example uses four despite a fuel of 4,267. Those values demonstrate conservatism of this formula in the included cases, not a worst-case efficiency theorem.

11 The cache invariant is proved for the executed update

The cached state stores a gain, one residual matrix per output label, their energies, and their total. The invariant links every stored quantity to the current gain's residual. It is an equation invariant, not an informal assertion that cached values remain approximately correct.

Komorebi/CachedPacketState.lean, lines 13–17.

```
structure State (ι κ α : Type*) where
  gain : Matrix ι κ ℚ
  residuals : α → Matrix ι κ ℚ
  energies : α → ℚ
  total : ℚ
```

Komorebi/CachedPacketState.lean, lines 33–36.

```
def Valid (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ) (c : State ι κ α) : Prop :=
  (∀ a, c.residuals a = Q a * M.residual c.gain) ∧
  (∀ a, c.energies a = rationalEnergy (c.residuals a)) ∧
  c.total = ∑ a, c.energies a
```

Komorebi/CachedPacketState.lean, lines 26–31.

```
def initialState (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (B : Matrix ι κ ℚ) : State ι κ α :=
  let F := M.residual B
  let G := fun a => Q a * F
  let e := fun a => rationalEnergy (G a)
  ⟨B,G,e,∑ a, e a⟩
```

Komorebi/CachedPacketState.lean, lines 43–49.

```
def step (K : Kernel κ) (a : α) (c : State ι κ α) : State ι κ α :=
  let G := c.residuals a * K.transition
  let e := rationalEnergy G
  ⟨c.gain+K.step • c.residuals a,
    fun b => if b=a then G else c.residuals b,
    fun b => if b=a then e else c.energies b,
    c.total-c.energies a+e⟩
```

Only the selected residual is multiplied by the precomputed transition. Only its energy is recomputed. The total changes by subtracting the old selected energy and adding the new one. The gain changes by the selected residual. The preservation proof uses the selected/unselected identities and a finite-sum difference identity.

Komorebi/CachedPacketState.lean, lines 66–91.

```
theorem step_valid (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (hQ : RationalPacketsValid Q) (c : State ι κ α) (hc : Valid M Q c) (a : α) :
    Valid M Q (step (prepare M) a c) := by
  refine ⟨?_,?_,?_⟩
  · intro b
    rw [step_gain M Q c hc a]
    by_cases hb : b=a
    · subst b
      simp only [step,ite_true,prepare,hc.1 a,selected_residual M Q hQ]
    · rw [step_other_residual _ _ _ hb,unselected_residual M Q hQ a b hb,hc.1 b]
  · intro b
    by_cases hb : b=a
    · subst b; simp only [step,ite_true]
    · simpa only [step,ite_eq_right hb] using hc.2.1 b
  · change c.total-c.energies a+rationalEnergy (c.residuals a*(prepare M).transition) =
      ∑ b, if b=a then rationalEnergy (c.residuals a*(prepare M).transition) else c.energies b
    rw [hc.2.2]
    have hi : (fun b => if b=a then rationalEnergy (c.residuals a*(prepare M).transition) else c.
    energies b) =
        (fun b => c.energies b + if b=a then rationalEnergy (c.residuals a*(prepare M).transition)-c.
    energies a else 0) := by
      funext b
      split_ifs with hb
      · subst b; ring
      · ring
    rw [hi,Finset.sum_add_distrib]
    simp only [Finset.sum_ite_eq',Finset.mem_univ,ite_true]
    ring
```

All of these updates are exact rationals. No rounding radius is hidden in `Valid`. This exactness is why the source can reuse untouched energies without carrying extra approximation errors.

12 Cached execution equals the recomputed greedy trajectory

The cached total equals the full residual energy by the packet partition, and its score equals the model's direct residual score. The cache-based maximum also equals the recomputed maximum. These facts connect an efficient state representation to the previously proved gain iteration.

Komorebi/CachedPacketState.lean, lines 103–107.

```
theorem score_eq (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (hQ : RationalPacketsValid Q) (c : State ι κ α) (hc : Valid M Q c) :
    score (prepare M) c = M.residualScore c.gain := by
  rw [score,total_eq_residual_energy M Q hQ c hc]
  simp only [prepare,RationalModel.residualScore,div_eq_mul_inv,mul_assoc]
```

Komorebi/CachedPacketTrajectory.lean, lines 27–32.

```
theorem chosen_eq (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (c : State ι κ α) (hc : Valid M Q c) (seed : α) (scan : List α) :
    chosen seed scan c = selectPacket M Q seed scan c.gain := by
  have he : c.energies = fun a => rationalEnergy (Q a*M.residual c.gain) := by
    funext a; rw [hc.2.1 a,hc.1 a]
  simp only [chosen,selectPacket,he]
```

Komorebi/CachedPacketTrajectory.lean, lines 53–60.

```
theorem trajectory_gain (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ)
    (hQ : RationalPacketsValid Q) (c : State ι κ α) (hc : Valid M Q c) (seed : α) (scan : List α) (n : ℕ
    ) :
    (trajectory (prepare M) seed scan c n).gain = greedyGain M Q seed scan c.gain n := by
  induction n with
  | zero => rfl
  | succ n ih =>
    rw [trajectory,advance_gain M Q _ (trajectory_valid M Q hQ c hc seed scan n),ih]
    rfl
```

The trajectory theorem is stronger than agreement at one exported final matrix. It proves equality for every natural update count from every valid initial cached state under valid output geometry. Consequently the cached solver inherits greedy convergence without treating a reference Python implementation as an oracle.

The origin theorem below further proves that a returned certificate gain came from an actual greedy iterate within the computed fuel. The final verifier cannot silently replace the proposed gain with an unrelated one.

Komorebi/CachedPacketOrigin.lean, lines 10–32.

```
theorem cached_solver_gain_origin (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (seed : α) (scan : List α) (ε : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ)
    (p : ℕ) (c : RationalCandidate ι κ α β) (hr : solveCached M Q D seed scan ε probes B = some (p,c)) :
    ∃ k ≤ adaptiveFuel M B ε (Fintype.card α), c.gain = greedyGain M Q seed scan B k := by
  unfold solveCached at hr
  split at hr
  next hg =>
    dsimp only at hr
    cases hsearch : search (prepare M) seed scan ε (adaptiveFuel M B ε (Fintype.card α)) (initialState M
    Q B) with
    | none => simp only [hsearch] at hr; contradiction
    | some out =>
      simp only [hsearch] at hr
      have hQ := ((checkSetup_eq_true M Q D).mp hg.2.1).2.1
      obtain ⟨k,hk,he⟩ := search_origin (prepare M) seed scan ε _ (initialState M Q B) out hsearch
      have hc := (certifyGain_origin M Q D ε probes out.gain p c hr).1
      refine ⟨k,hk,?_⟩
      calc
        c.gain = out.gain := by rw [hc]; rfl
        _ = greedyGain M Q seed scan B k := by
          rw [he,trajectory_gain M Q hQ _ (initialState_valid M Q B)]
          rfl
  next => contradiction
```

13 Search completeness closes the algorithmic existence step

The bounded search inspects the initial state's exact score, then advances until it passes or fuel is exhausted. A zero-fuel call still checks its current state. Soundness and completeness are separate induction arguments: a successful result satisfies the threshold, and a trajectory whose final permitted state passes guarantees some early-stopping result.

Komorebi/CachedPacketSearch.lean, lines 10–12.

```
def search (K : Kernel κ) (seed : α) (scan : List α) (ε : ℚ) : ℕ → State ι κ α → Option (State ι κ α)
  | 0,c => if score K c ≤ ε^2/4 then some c else none
  | n+1,c => if score K c ≤ ε^2/4 then some c else search K seed scan ε n (advance K seed scan c)
```

Komorebi/CachedPacketSearch.lean, lines 45–58.

```
theorem search_complete (K : Kernel κ) (seed : α) (scan : List α) (ε : ℚ)
    (n : ℕ) (c : State ι κ α) (hp : score K (trajectory K seed scan c n) ≤ ε^2/4) :
    ∃ out, search K seed scan ε n c = some out := by
  induction n generalizing c with
  | zero =>
    change score K c ≤ ε^2/4 at hp
    exact ⟨c,by rw [search,ite_eq_left hp]⟩
  | succ n ih =>
    by_cases hs : score K c ≤ ε^2/4
    · exact ⟨c,by rw [search,ite_eq_left hs]⟩
    · have hp' : score K (trajectory K seed scan (advance K seed scan c) n) ≤ ε^2/4 := by
        simpa only [trajectory_shift] using hp
      obtain ⟨out,hr⟩ := ih _ hp'
      exact ⟨out,by rw [search,ite_eq_right hs]; exact hr⟩
```

Komorebi/CachedPacketSolver.lean, lines 14–22.

```
def solveCached (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (seed : α) (scan : List α) (ε : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ) : Option (ℕ ×
    RationalCandidate ι κ α β) :=
  if 0 < ε ∧ checkSetup M Q D = true ∧ ∀ a, a ∈ scan then
    let K := prepare M
    match search K seed scan ε (adaptiveFuel M B ε (Fintype.card α)) (initialState M Q B) with
    | none => none
    | some c => certifyGain M Q D ε probes c.gain
  else none
```

The entry guard checks positive posterior tolerance, setup validity, and scan coverage. `prepare` computes the model step, innovation transition, and score weight once. The search begins from the actual cache constructor, whose validity is immediate. A successful cached state proceeds to independent rounded verification of its gain.

The solver termination proof composes established obligations rather than assuming convergence of an arbitrary cache: valid initialization, validity preservation, exact trajectory equivalence, geometric fuel sufficiency, search completeness, and final-certifier completeness.

Komorebi/CachedPacketSolver.lean, lines 37–60.

```
theorem solveCached_terminates (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (hQ : RationalPacketsValid Q) (hD : RationalPacketsValid D)
    (seed : α) (scan : List α) (hscan : ∀ a, a ∈ scan)
    (ε : ℚ) (hε : 0 < ε) (probes : List ℕ) (B : Matrix ι κ ℚ) :
    ∃ p c, solveCached M Q D seed scan ε probes B = some (p,c) ∧
      checkCandidate M Q D c = true ∧ checkTolerance M c ε = true := by
  let init := initialState M Q B
  let fuel := adaptiveFuel M B ε (Fintype.card α)
  have hv : Valid M Q init := initialState_valid M Q B
  have hp := trajectory_score_at_fuel M hs Q hQ init hv seed scan hscan ε hε fuel le_rfl
```

```
  obtain ⟨out,hr⟩ := search_complete (prepare M) seed scan ε fuel init hp
  obtain ⟨k,_,he⟩ := search_origin (prepare M) seed scan ε fuel init out hr
  have ho : Valid M Q out := by rw [he]; exact trajectory_valid M Q hQ init hv seed scan k
  have hb : M.residualScore out.gain ≤ ε^2/4 := by
    rw [← score_eq M Q hQ out ho]
    exact search_score (prepare M) seed scan ε fuel init out hr
  obtain ⟨p,c,hc⟩ := certifyGain_complete M hs Q D hQ hD ε hε probes out.gain hb
  have hg := (checkSetup_eq_true M Q D).mpr ⟨hs,hQ,hD⟩
  have hout : solveCached M Q D seed scan ε probes B = some (p,c) := by
    rw [solveCached,ite_eq_left ⟨hε,hg,hscan⟩]
    change (match search (prepare M) seed scan ε fuel init with | none => none | some x => certifyGain M
    Q D ε probes x.gain) = _
    rw [hr]; exact hc
  exact ⟨p,c,hout,solveCached_sound M Q D seed scan ε probes B p c hout⟩
```

14 Outward dyadic arithmetic has its own soundness proof

A scalar ball is a rational center and absolute radius. Enclosure means $|x - c| \le r$. Its radius is not enforced by the record constructor alone. A proof of enclosure implies nonnegativity, and arithmetic lemmas propagate that proof through addition, multiplication, and finite sums.

Komorebi/RationalBall.lean, lines 11–14.

```
structure Ball where
  center : ℚ
  radius : ℚ
  deriving DecidableEq, Repr
```

Komorebi/RationalBall.lean, lines 16–16.

```
def Encloses (b : Ball) (x : ℚ) : Prop := |x - b.center| ≤ b.radius
```

Komorebi/DyadicBall.lean, lines 22–24.

```
def round (p : ℕ) (b : Ball) : Ball :=
  let c := down p b.center
  ⟨c, up p (b.radius + |b.center-c|)⟩
```

Komorebi/DyadicBall.lean, lines 26–33.

```
theorem round_encloses (p : ℕ) {b : Ball} {x : ℚ} (h : Encloses b x) :
    Encloses (round p b) x := by
  change |x-down p b.center| ≤ up p (b.radius+|b.center-down p b.center|)
  calc
    _ = |(x-b.center)+(b.center-down p b.center)| := by congr 1; ring
    _ ≤ |x-b.center|+|b.center-down p b.center| := abs_add_le _ _
    _ ≤ b.radius+|b.center-down p b.center| := add_le_add h le_rfl
    _ ≤ _ := le_up _ _
```

At fractional precision p, centers are rounded down to a multiple of 2^{-p}. The radius is first inflated by the displacement of the center, then rounded up. This preserves enclosure for both positive and negative quantities. Integer parts are unbounded in this mathematical implementation, so the theorem is not an IEEE floating-point overflow guarantee.

The final candidate retains the gain unchanged. It evaluates each residual packet and innovation vector with outward arithmetic, then extracts conservative energy upper bounds. It is not a rounded gain

update.

Komorebi/RoundedCandidate.lean, lines 11–16.

```
def roundedCandidate (p : ℕ) (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ) (B : Matrix ι κ ℚ) :
    RationalCandidate ι κ α β where
  gain := B
  packetUpper a b := matrixEnergyUpper (packetBall p M B (Q a) (D b))
  innovationUpper := vectorEnergyUpper (innovationVectorBall p M)
```

Komorebi/RoundedCandidate.lean, lines 25–32.

```
theorem roundedCandidate_valid (p : ℕ) (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ) (B : Matrix ι κ ℚ)
    (hs : 0 < M.noiseFloor) (hQ : RationalPacketsValid Q) (hD : RationalPacketsValid D) :
    ValidCandidate M Q D (roundedCandidate p M Q D B) := by
  refine ⟨hs, hQ, hD, ?_, ?_⟩
  · intro a b
    exact matrixEnergyUpper_sound (packetBall_encloses p M B (Q a) (D b))
  · exact vectorEnergyUpper_sound (innovationVectorBall_encloses p M)
```

15 Finite verification precision is computed, not guessed

Magnitude and radius budgets bound score inflation by a computable coefficient times the dyadic mesh. The sufficient-precision function chooses a precision for a positive slack. Every precision above that threshold has the claimed inflation bound. Individual finite-precision scores need not be monotone for this theorem to work.

Komorebi/PrecisionCompleteness.lean, lines 13–15.

```
def sufficientPrecision (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ) (B : Matrix ι κ ℚ) (δ : ℚ) : ℕ :=
  precisionFor (scoreRoundCoeff M Q D B) δ
```

Komorebi/PrecisionCompleteness.lean, lines 19–26.

```
theorem rounded_score_le_with_slack (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ) (B : Matrix ι κ ℚ)
    (hs : 0 < M.noiseFloor) (δ : ℚ) (hδ : 0 < δ) (p : ℕ)
    (hp : sufficientPrecision M Q D B δ ≤ p) :
    candidateScore M (roundedCandidate p M Q D B) ≤
      candidateScore M (exactCandidate M Q D B)+δ := by
  have hc := coefficient_mesh_le (scoreRoundCoeff M Q D B) δ hδ p hp
  exact (rounded_score_inflation p M Q D B hs).trans (add_le_add le_rfl hc)
```

Komorebi/RoundedSolverSearch.lean, lines 12–14.

```
def certifyGain (M : RationalModel ι κ υ ω) (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (ε : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ) : Option (ℕ × RationalCandidate ι κ α β) :=
  certifyTagged (probes++[sufficientPrecision M Q D B (ε^2/4)]) M Q D B ε
```

Komorebi/RoundedSolverSearch.lean, lines 25–37.

```
theorem certifyGain_complete (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (hQ : RationalPacketsValid Q) (hD : RationalPacketsValid D)
    (ε : ℚ) (hε : 0 < ε) (probes : List ℕ) (B : Matrix ι κ ℚ)
    (hscore : M.residualScore B ≤ ε^2/4) :
```

```
    ∃ p c, certifyGain M Q D ε probes B = some (p,c) := by
  let p := sufficientPrecision M Q D B (ε^2/4)
  have hm : candidateScore M (exactCandidate M Q D B)+ε^2/4 ≤ ε^2 := by
    rw [exactCandidate_score M Q D hQ hD]
    nlinarith only [hscore,sq_nonneg ε]
  have hr := certifyRounded_of_slack M Q D B hs hQ hD ε (ε^2/4) hε.le
    (by positivity) hm p le_rfl
  exact certifyTagged_complete _ p (by simp [p]) M Q D B ε _ hr
```

The caller's finite probes are tried first and can find a cheap passing precision. The sufficient precision is appended regardless of whether the probe list is empty or contains repeated entries. The fallback is a theorem-backed finite completion mechanism for a gain with sufficient exact score slack.

A failed rounded check does not prove that the true posterior error is too large. Conservative bounds can fail while an exact or higher-precision bound would pass. A successful check, however, is sound. This asymmetry is part of the interface and prevents interpreting a verifier rejection as a statistical impossibility result.

16 Output radius and posterior tolerance mean different things

For the returned gain B, the approximate Gaussian has exact rational mean $m + B(y - Hm)$ and Joseph covariance

$$(I - BH)P(I - BH)^{\mathsf{T}} + BRB^{\mathsf{T}}.$$

The parameter graph encloses those entries. It does not directly enclose the exact posterior's coordinates merely because the law is close in W_2.

Komorebi/PosteriorOutput.lean, lines 13–15.

```
def outputMean [Fintype κ] [Fintype ι]
    (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) : ι → ℚ :=
  M.mean + B *ᵥ M.innovationVector
```

Komorebi/PosteriorOutput.lean, lines 21–25.

```
def outputCovariance [Fintype ι] [Fintype κ] [Fintype υ] [Fintype ω]
    [DecidableEq ι] [DecidableEq κ]
    (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) : Matrix ι ι ℚ :=
  outputOperator M B * M.prior * (outputOperator M B).transpose +
    B * M.noise * B.transpose
```

Komorebi/PosteriorOutputPrecision.lean, lines 14–15.

```
def outputPrecision (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) (δ : ℚ) : ℕ :=
  precisionFor (outputCoefficient M B) δ
```

Komorebi/PosteriorOutputPrecision.lean, lines 56–59.

```
theorem represent_spec (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ)
    (δ : ℚ) (hδ : 0 < δ) :
    EnclosesParameters M B (represent M B δ) ∧ RadiiAtMost (represent M B δ) δ :=
  ⟨evaluate_encloses _ M B, evaluate_radii_le M B δ hδ _ le_rfl⟩
```

Komorebi/PosteriorOutputSemantics.lean, lines 36–40.

```
theorem approximation_eq_output_gaussian (M : RationalModel ι κ υ ω) (B : Matrix ι κ ℚ) :
    M.approximation B =
      multivariateGaussian (realVector (outputMean M B)) (realMatrix (outputCovariance M B)) := by
  rw [outputMean_real, outputCovariance_real]
  rfl
```

Here ε controls the target-to-approximation transport distance, and δ controls every reported interval radius around the approximation's exact parameters. They are independently supplied tolerances and have different codomains and purposes. Shrinking δ does not change the selected gain. Shrinking ε can require more gain updates.

The approximation covariance is positive semidefinite by the Joseph construction under the checked model hypotheses. Independent rounding of its entries to interval centers need not preserve every covariance constraint. The enclosed exact covariance remains the semantic object certified by the result.

17 Execution example I: a rectangular coordinate model

The first complete execution has two state coordinates and one observed coordinate. The prior covariance is $\mathrm{diag}(4,1)$, the observation is the sum of the two coordinates, noise variance is one, and the prior mean is zero. The exact gain is $(2/3, 1/6)^{\mathsf{T}}$. Coordinate output packets form a complete pair and the input packet is the one-dimensional identity.

Komorebi/GeneratedCachedCoordinate.lean, lines 14–20.

```
def model : RationalModel (Fin 2) (Fin 1) (Fin 2) (Fin 1) where
  observationMatrix := !![1, 1]
  priorFactor := !![2, 0; 0, 1]
  noiseFactor := !![0]
  noiseFloor := 1
  mean := ![0, 0]
  observation := ![1]
```

Komorebi/GeneratedCachedCoordinate.lean, lines 21–21.

```
def outputs : Fin 2 → Matrix (Fin 2) (Fin 2) ℚ := ![!![1, 0; 0, 0], !![0, 0; 0, 1]]
```

Komorebi/GeneratedCachedCoordinate.lean, lines 22–22.

```
def inputs : Fin 1 → Matrix (Fin 1) (Fin 1) ℚ := ![!![1]]
```

Komorebi/GeneratedCachedCoordinate.lean, lines 24–24.

```
def finalGain : Matrix (Fin 2) (Fin 1) ℚ := !![(228/343); (57/343)]
```

Starting from the zero gain, the executed output labels are $0, 1, 0, 1, 0, 1$. The innovation variance is six, the chosen trace-cap step is $1/7$, and three visits to each packet give $B = (228/343, 57/343)^{\mathsf{T}}$. This observation operator is rectangular, so no theorem can be relying on an inverse observation matrix.

The posterior tolerance is $1/20$, the output radius cap is $1/1000$, the passing certificate precision is 12, and parameter precision is 32. The exact score $34/117649$ passes the stricter iteration threshold $\varepsilon^2/4 = 1/1600$. The outward certificate score $481/1048576$ passes $\varepsilon^2 = 1/400$.

Komorebi/GeneratedCachedCoordinate.lean, lines 32–32.

```
theorem cached_return : solveCached model outputs inputs 0 [0, 1] (1/20) [4, 12] initial = some (12,
    certificate) := by decide +kernel
```

Komorebi/GeneratedCachedCoordinate.lean, lines 40–40.

```
theorem output_precision_computed : outputPrecision model finalGain (1/1000) = 32 := by decide +kernel
```

Komorebi/GeneratedCachedCoordinate.lean, lines 41–41.

```
theorem mean_centers_computed : (fun i => (outputArrays.mean i).center) = ![(2854963683/4294967296),
    (89217615/536870912)] := by decide +kernel
```

Komorebi/GeneratedCachedCoordinate.lean, lines 42–42.

```
theorem mean_radii_computed : (fun i => (outputArrays.mean i).radius) = ![(1/4294967296), (1/4294967296)
    ] := by decide +kernel
```

Komorebi/GeneratedCachedCoordinate.lean, lines 44–44.

```
theorem covariance_radii_computed : (fun i j => (outputArrays.covariance i j).radius) = !![(1/536870912),
     (7/4294967296); (7/4294967296), (1/536870912)] := by funext i j; fin_cases i <;> fin_cases j <;>
    decide +kernel
```

These values are not merely computed outside Lean and compared to a manually written certificate. The explicit module checks the actual `runCached` return and obtains the universal output contract by successful-return soundness.

Komorebi/GeneratedCachedCoordinate.lean, lines 46–48.

```
theorem output_return : runCached model outputs inputs 0 [0, 1] (1/20) (1/1000) [4, 12] initial = some
    outputResult := by
  simp only [runCached,ite_eq_left (show (0 : ℚ) < (1/1000) by norm_num),cached_return]
  rfl
```

Komorebi/GeneratedCachedCoordinate.lean, lines 49–50.

```
theorem output_valid : ResultValid model outputs inputs (1/20) (1/1000) outputResult :=
  runCached_sound model outputs inputs 0 [0, 1] (1/20) (1/1000) [4, 12] initial outputResult
    output_return
```

18 Execution example II: a singular prior and rotated packets

The second execution has $H = I_2$, $P = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, $R = I_2$, and $y = (1, -1)$. The prior factor has only one column. Its rank-one prior is supported on the common-mode line, while the observation points in the difference direction. The common-mode and difference-mode rational projectors give a rotated complete packet family.

Komorebi/GeneratedCachedRotated.lean, lines 14–20.

```
def model : RationalModel (Fin 2) (Fin 2) (Fin 1) (Fin 1) where
  observationMatrix := !![1, 0; 0, 1]
  priorFactor := !![1; 1]
  noiseFactor := !![0; 0]
  noiseFloor := 1
  mean := ![0, 0]
  observation := ![1, -1]
```

Komorebi/GeneratedCachedRotated.lean, lines 21–21.

```
def outputs : Fin 2 → Matrix (Fin 2) (Fin 2) ℚ := ![![(1/2), (1/2); (1/2), (1/2)], ![(1/2), (-1/2);
    (-1/2), (1/2)]]
```

Komorebi/GeneratedCachedRotated.lean, lines 22–22.

```
def inputs : Fin 2 → Matrix (Fin 2) (Fin 2) ℚ := ![![(1/2), (1/2); (1/2), (1/2)], ![(1/2), (-1/2);
    (-1/2), (1/2)]]
```

Komorebi/GeneratedCachedRotated.lean, lines 24–24.

```
def finalGain : Matrix (Fin 2) (Fin 2) ℚ := ![![(203/625), (203/625); (203/625), (203/625)]
```

All four selected labels are zero: the residual lies in the common-mode packet, and the difference packet has zero residual energy. The final gain has every entry $203/625$. Its exact score is $3072/390625$ and its rounded certificate score is $9111/1048576$. With $\varepsilon = 1/4$, they pass $1/64$ and $1/16$ respectively.

Both the exact posterior and approximation mean are zero in this case. The exact posterior covariance has every entry $1/3$. The final approximate Joseph covariance has every entry $130379/390625$. That distinction remains certified by the law guarantee even though the selected observation produces matching means.

Komorebi/GeneratedCachedRotated.lean, lines 32–32.

```
theorem cached_return : solveCached model outputs inputs 0 [0, 1] (1/4) [4, 12] initial = some (12,
    certificate) := by decide +kernel
```

Komorebi/GeneratedCachedRotated.lean, lines 41–41.

```
theorem covariance_centers_computed : (fun i j => (outputArrays.covariance i j).center) =
    ![![(179191533/536870912), (179191533/536870912); (179191533/536870912), (179191533/536870912)] := by
     funext i j; fin_cases i <;> fin_cases j <;> decide +kernel
```

Komorebi/GeneratedCachedRotated.lean, lines 42–42.

```
theorem covariance_radii_computed : (fun i j => (outputArrays.covariance i j).radius) = ![(7/4294967296)
    , (7/4294967296); (7/4294967296), (7/4294967296)] := by funext i j; fin_cases i <;> fin_cases j <;>
    decide +kernel
```

Komorebi/GeneratedCachedRotated.lean, lines 44–46.

```
theorem output_return : runCached model outputs inputs 0 [0, 1] (1/4) (1/1000) [4, 12] initial = some
    outputResult := by
  simp only [runCached,ite_eq_left (show (0 : ℚ) < (1/1000) by norm_num),cached_return]
  rfl
```

Komorebi/GeneratedCachedRotated.lean, lines 47–48.

```
theorem output_valid : ResultValid model outputs inputs (1/4) (1/1000) outputResult :=
  runCached_sound model outputs inputs 0 [0, 1] (1/4) (1/1000) [4, 12] initial outputResult
    output_return
```

The returned intervals are very narrow compared with the supplied radius cap. This does not make the approximate covariance equal to the exact posterior covariance. It means the finite output graph represents the approximate covariance accurately. The W_2 theorem supplies the separate target comparison.

19 A checked counterexample prevents a false rounded-cache claim

The exact invariant cannot survive independent rounding of gain and residual centers without an additional error budget. The archived scalar example has prior variance one, observation coefficient one, noise variance one, observed value one, and zero initial gain. Its step is $1/3$. The ideal gain after one step is $1/3$, and the ideal residual is also $1/3$.

Rounding those two centers down independently to precision zero produces stored gain zero and stored residual zero. The true residual of the stored gain is one. Therefore the naive cached score reports zero, while the true score is two. This is an executed counterexample with valid setup, rather than a hypothetical concern about roundoff.

Komorebi/CachedRoundingBoundary.lean, lines 20–25.

```
def naiveRounded : State (Fin 1) (Fin 1) (Fin 1) :=
  let ideal := step (prepare model) 0 (initialState model packets zeroGain)
  let B := fun i j => down 0 (ideal.gain i j)
  let G := fun a i j => down 0 (ideal.residuals a i j)
  let e := fun a => rationalEnergy (G a)
  ⟨B,G,e,∑ a, e a⟩
```

Komorebi/CachedRoundingBoundary.lean, lines 27–27.

```
theorem naive_score_zero : score (prepare model) naiveRounded = 0 := by decide +kernel
```

Komorebi/CachedRoundingBoundary.lean, lines 28–28.

```
theorem true_score_two : model.residualScore naiveRounded.gain = 2 := by decide +kernel
```

Komorebi/CachedRoundingBoundary.lean, lines 29–29.

```
theorem naive_cache_invalid : ¬ Valid model packets naiveRounded := by unfold Valid; decide +kernel
```

Komorebi/CachedRoundingBoundary.lean, lines 32–33.

```
theorem naive_final_verification_rejected :
    Komorebi.RoundoffSolver.certifyGain model packets packets (1/10) [12] naiveRounded.gain = none := by
      decide +kernel
```

The independent final verifier evaluates the actual stored gain and rejects the false convergence signal for tolerance $1/10$. This gives a precise division of responsibilities: an internal heuristic or invalid cache can be misleading, but its claim does not bypass final certification. A future rounded selective cache needs a propagated cache-error invariant and its own termination proof. The existing exact invariant must not be relabeled as one.

20 Negative controls test specific failure boundaries

The archive contains three intentionally invalid Lean controls. They were expected to fail, and their successful rejection is preserved separately from successful mathematical modules. A false rational equality tests ordinary proof rejection. A placeholder tests the warnings-as-errors route. A dropped radius tests that a zero-radius interval at zero cannot enclose a distinct rational.

Control	Obligation intentionally violated	Archived exit
`FalseEquality`	An explicit false equality	1
`Placeholder`	An admitted placeholder under warning rejection	1
`DroppedRoundingRadius`	Enclosure of 1/1024 by center zero and radius zero	1

The radius control is copied here as a complete intentionally failing file fragment. It is not a proof excerpt from an accepted module.

```
import Komorebi.RationalBall
example : Komorebi.RoundedPosterior.Encloses ⟨0,0⟩ (1/1024) := by
  unfold Komorebi.RoundedPosterior.Encloses
  decide +kernel
```

The expected result is rejection. A tool report that only says the command completed would be ambiguous here. The archived JSON records nonzero exit status, `expected_rejection=true`, source hashes, and log hashes for each control.

Producer-side tests also reject malformed or unsupported inputs. They exercise parser policy and resource handling, while the Lean controls exercise proof and enclosure boundaries. Neither set proves a universal refinement theorem for the Python parser. An exported successful instance certifies the literal model and data appearing in its emitted Lean source.

21 The rounded full-gain branch remains a separate result

The source also retains a solver in which failed verification is followed by a genuinely rounded full-gain update. Its convergence analysis tracks update error and chooses an update accuracy plus finite fuel. It is not the cached selective program with rounding switched on.

`Komorebi/RoundedSolverSearch.lean`, lines 92–97.

```
def solveRounded (M : RationalModel ι κ υ ω)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (ε : ℚ) (probes : List ℕ) (B : Matrix ι κ ℚ) : Option (ℕ × RationalCandidate ι κ α β) :=
  if 0 < ε ∧ checkSetup M Q D = true then
    solveRoundedFuel M Q D ε probes (updateAccuracy M ε) (roundedFuel M B ε) B
  else none
```

`Komorebi/RoundedSolverSearch.lean`, lines 111–124.

```
theorem solveRounded_terminates (M : RationalModel ι κ υ ω) (hs : 0 < M.noiseFloor)
    (Q : α → Matrix ι ι ℚ) (D : β → Matrix κ κ ℚ)
    (hQ : RationalPacketsValid Q) (hD : RationalPacketsValid D)
    (ε : ℚ) (hε : 0 < ε) (probes : List ℕ) (B : Matrix ι κ ℚ) :
    ∃ p c, solveRounded M Q D ε probes B = some (p,c) ∧
      checkCandidate M Q D c = true ∧ checkTolerance M c ε = true := by
  have hscore := rounded_score_at_fuel M hs ε hε (updateAccuracy M ε) le_rfl
    B (roundedFuel M B ε) le_rfl
  have hp := certifyGain_complete M hs Q D hQ hD ε hε probes _ hscore
  obtain ⟨p,c,hr⟩ := solveRoundedFuel_complete M Q D ε probes _ _ B hp
  have hs0 := (checkSetup_eq_true M Q D).mpr ⟨hs,hQ,hD⟩
  have hout : solveRounded M Q D ε probes B = some (p,c) := by
    simpa only [solveRounded,hε,hs0,and_self,ite_true] using hr
  exact ⟨p,c,hout,solveRounded_sound M Q D ε probes B p c hout⟩
```

This theorem confirms that the retained full-update branch also guarantees a successful certified candidate for positive tolerance under admissible model and packet hypotheses. Its proof invokes the rounded-score-at-fuel theorem, final verification completeness, and bounded-search completeness. The distinction matters when a reader asks whether the release has finite-precision updates: yes for this full branch, while selective cached gain updates in the canonical branch remain exact rationals.

The seven-case bounded benchmark compares full Richardson, recomputed greedy selection, and cached greedy selection. The 21 benchmark modules certify static explicit final candidates. The two generated modules separately certify complete canonical program executions. Universal algorithm theorems and those two kinds of instance evidence should be counted separately.

22 Rebuild the pinned project without overwriting the archive

The source archive preserves the exact release and failures. A new rebuild should use a working copy, because the original rebuild script deliberately regenerates active compiled caches, audit records, seals, and result files. The following commands are a reproducible route, not a claim that they were executed during this publication session.

Restore the separate Linux x86_64 portable runtime, then confirm Lean 4.34.0-rc2 at commit `6a10ac8c22beadecabdbb0919c2b50214762f91d` and mathlib at `aeb72f1e9981b6d35b426b16dca7759be3e7ba06`. The source package contains restoration metadata and starter information, but not the large runtime payload parts. Its metadata describes a Linux runtime intended also for x86_64 WSL2, rather than a native Windows compiler.

```
# Run from the publication directory, after extracting the source archive.
cp -a source/FD_Certified_Inference_Core_v1.0.0_Checked rebuild_work
cd rebuild_work
python3 checks/rebuild_release.py \
  --runtime /path/to/Komorebi_Portable_Lean_4.34.0-rc2
```

The script compiles modules in `MODULE_ORDER.json`, uses one Lean thread and warnings as errors, runs negative controls and exact cache diagnostics, verifies the release seal, and reconciles benchmark reports. It removes active project caches for a clean project-source rebuild while reusing the pinned mathlib caches. This is not a clean rebuild of every external dependency from source.

For a new proof-producing Python example, emit a new module in the working copy, regenerate its inventory, and rebuild:

```
python3 checks/cached_posterior.py examples/adaptive_coordinate.json \
  --output results/local_cached.json \
  --emit-lean Komorebi/LocalCached.lean \
  --namespace LocalCached --try-precisions 4 12
python3 checks/build_inventory.py
python3 checks/rebuild_release.py \
  --runtime /path/to/Komorebi_Portable_Lean_4.34.0-rc2
```

Treat `PASS_CACHED_PROVISIONAL` as provisional until the emitted module compiles. A reported `RESOURCE_LIMIT` is a bounded producer failure, not a refutation of mathematical finite termination. The default producer caps include dimension eight, 10,000 updates, rational bit limits, precision limits, scan limits, and input byte limits.

The publication's read-only check is narrower and can run without Lean:

```
python3 scripts/lean_inventory.py
```

It rechecks all 939 manifest entries, all 117 final active source/snapshot/log receipts, 288 direct project dependency records, 543 declaration locations and axiom reports, and every source-attributed listing in this companion. These source consistency results are fresh checks. The September kernel runs remain archived evidence.

23 How to extend the proof without weakening the contract

A useful extension begins by choosing which invariant changes. Sequential filtering changes the target model over a sequence of observations and needs an explicit propagated state/posterior contract. Uncertain input data changes exact model semantics and requires an uncertainty interpretation. Rounded selective caching changes the exact residual equations and requires error bounds coupled to selection and convergence. A changing packet hierarchy changes the geometry that underpins energy completeness.

The present theorem chain gives concrete reuse points. Gaussian disintegration and residual transport estimates can support new candidate producers. The finite checker can certify any explicitly represented gain whose rational upper bounds pass. Output representation can enclose the Joseph parameters of any supplied rational gain. What cannot be reused unchanged is a trajectory termination proof when updates, selection, or cache equations have changed.

The release's current accomplishment is an integrated algorithmic theorem: within its declared input class, its own program returns a valid result, and every successful return has that result contract. Scope boundaries identify future work rather than leaving this scoped theorem unfinished. They also prevent a proof of one Gaussian update from being reported as a universal sequential Bayesian inference system.

A Module order and declaration map

The table follows the archived compilation order. Counts are named theorem/lemma declarations in `DECLARATIONS.json`; definitions and structures are not included. Support lemmas, concrete checks, and benchmark statements are counted together, so the total is not a count of independent discoveries. `ReleaseAxiomAudit` imports the entire project and prints axiom dependencies rather than introducing new mathematical declarations.

Order	Module	Count
1	`RationalMatrixCertificates`	12
2	`PacketMatrixEnergy`	6
3	`SelfAdjointPackets`	4
4	`RationalPacketCertificates`	5
5	`RationalCertificateChecker`	6
6	`RichardsonIteration`	6
7	`GeometricFuel`	3
8	`RationalRichardsonDefs`	5
9	`FDGaussianTransport`	5
10	`FDMatrixOperatorCompat`	5
11	`FDMatrixEnergyBounds`	5
12	`FDKalmanDefs`	0
13	`FDInnovationIdentities`	8
14	`FDInnovationConditioning`	4
15	`GramNoiseFloor`	3
16	`PacketEnergyDecomposition`	5
17	`PacketVectorEnergy`	5
18	`PacketEnergyRefinement`	2
19	`PacketCoarsening`	4
20	`FDMatrixPacketEnergy`	5
21	`FDGaussianSum`	3
22	`FDGaussianQuadraticMoment`	4
23	`QuadraticTransport`	5
24	`FDGaussianCoupling`	6
25	`FDJointCovQuadrants`	4
26	`FDJointCovAssembly`	1
27	`FDPosteriorCovPSD`	2
28	`FDInexactGainAlgebra`	9
29	`FDGaussianKernel`	2
30	`FDJointPacking`	6
31	`FDJointCovPSD`	2
32	`FDJointLiftAlgebra`	7
33	`FDForwardJointLaw`	1
34	`FDPosteriorKernel`	1
35	`FDReverseJointLaw`	1
36	`FDGaussianConditioning`	5
37	`FDGainSampler`	6

Order	Module	Count
38	FDInexactGainCoupling	5
39	QuadraticTransportBounds	2
40	FDResidualEnergyCertificate	3
41	FDResidualPacketCertificate	4
42	RationalModelSemantics	8
43	RichardsonSpectrum	6
44	RichardsonEnergy	5
45	RationalRichardsonGeometry	8
46	PacketSelectiveAlgebra	7
47	GreedyPacketSelection	6
48	RationalCandidateAcceptance	7
49	RationalRichardsonConvergence	11
50	GreedyPacketConvergence	4
51	RationalBall	9
52	DyadicBall	10
53	DyadicPrecision	5
54	PrecisionBudget	6
55	PrecisionEnergy	3
56	RoundedMatrix	11
57	PrecisionMatrix	16
58	RoundedModel	6
59	PrecisionModel	6
60	PrecisionSelection	1
61	RoundoffEnergy	7
62	RoundedGainEvaluation	6
63	RoundoffTracking	6
64	RoundoffConvergence	11
65	PrecisionScoreAlgebra	1
66	RoundedCandidate	5
67	PrecisionScore	6
68	PosteriorEnclosureBounds	1
69	RationalPosteriorCertificate	6
70	RoundedPosteriorSound	3
71	RoundedPrecisionSearch	2
72	PrecisionCompleteness	5
73	TaggedPrecisionSearch	2
74	RoundedSolverSearch	7
75	RoundedSolverOrigin	3
76	AdaptivePacketSolver	4
77	AdaptivePacketInvariants	2
78	AdaptivePacketOrigin	2
79	AdaptivePacketPosterior	2
80	PosteriorOutput	5
81	PosteriorOutputPrecision	4
82	PosteriorOutputSemantics	5

Order	Module	Count
83	RoundedSolverPosterior	3
84	PosteriorOutputSolver	3
85	AdaptivePacketOutput	3
86	Benchmark0Cached	3
87	Benchmark0Full	3
88	Benchmark0Recompute	3
89	Benchmark1Cached	3
90	Benchmark1Full	3
91	Benchmark1Recompute	3
92	Benchmark2Cached	3
93	Benchmark2Full	3
94	Benchmark2Recompute	3
95	Benchmark3Cached	3
96	Benchmark3Full	3
97	Benchmark3Recompute	3
98	Benchmark4Cached	3
99	Benchmark4Full	3
100	Benchmark4Recompute	3
101	Benchmark5Cached	3
102	Benchmark5Full	3
103	Benchmark5Recompute	3
104	Benchmark6Cached	3
105	Benchmark6Full	3
106	Benchmark6Recompute	3
107	CachedPacketState	7
108	CachedPacketTrajectory	7
109	CachedPacketSearch	3
110	CachedPacketSolver	4
111	CachedPacketOrigin	1
112	CachedPacketOutput	3
113	RationalSolverDecidable	0
114	CachedRoundingBoundary	5
115	GeneratedCachedCoordinate	18
116	GeneratedCachedRotated	16
117	ReleaseAxiomAudit	0

B Evidence that accompanies the source

Artifact	Role
`RELEASE_SCOPE.md`	The input/output finish line and named extensions.
`PROOF_NOTE.md`	The original mathematical proof route.
`DECLARATIONS.json`	All 543 named theorem/lemma source locations.
`AXIOM_AUDIT.json`	Complete named-declaration coverage and permitted axiom dependencies.
`FINAL_VERIFICATION_RECEIPT.json`	Archived final successful builds, failed attempts, environment pins, controls and diagnostic summary.
`MANIFEST_SHA256.txt`	939 archived file digests.
`audit/`	Attempted-source snapshots, command receipts and exact process logs, including failed attempts.
`results/FORMAL_INSTANCES.json`	The two complete cached execution instances and their scopes.
`BENCHMARK_REPORT.md`	Controlled reference costs, timing methodology and limits.
`FAILURE_LOG.md`	Preserved failures and corrections.
`SELF_REVIEW.md`	Original internal adversarial review, with no external independence claim.
`RUNTIME_METADATA/`	Separate runtime restoration metadata and pins.
`reviews/lean_inventory.json`	Fresh publication-side byte, location, receipt, import and excerpt consistency results.

The archived axiom union is `Classical.choice`, `Quot.sound`, and `propext`. The check does not remove those foundations, prove Lean's kernel correct, or establish implementation refinement for all outside tools. It identifies the axioms reported for these named declarations and confirms that no added project axiom appears in that audit.

Archived final-module process time sums to 558.439 seconds. This is the sum of measured process durations, not an independently measured end-to-end wall-clock rebuild. The same distinction applies to hardware performance claims: verified mathematical results remain valid while bounded timings answer only their recorded experimental question.

The source, actual theorem contracts, execution-instance modules, failed controls, and read-only inventory make the publication reviewable at every major implication. A reader can begin at the output theorem, trace its obligations backward, and rerun the pinned working-copy build when the separate runtime is available.