The Proof as the Machine Checked It: The Verified Lean Formalization of the Navier–Stokes Blowup Argument

Bevan Hunt

DOI: PENDING CC BY 4.0

Self-published, no affiliation. Released under the Creative Commons Attribution 4.0 International License (creativecommons.org/licenses/by/4.0).

repository: https://github.com/truemagic-coder/ns-proof

Abstract

The OpenAI manuscript "Finite Time Blowup for Navier–Stokes" ships a Lean 4 formalization of its theorem, and an independent verification service returned a fully green verdict for it: the fork builds, both Navier–Stokes and both Euler target theorems re-check under two independent Lean kernels, and the axiom audit is clean. This paper documents exactly what that verdict is: the Lean statements the kernel proves, the two pieces of the argument we formalized ourselves – the manuscript's (8.19) torus-inverse estimate, which the kernel accepted as written, and the repaired $(10.19)'$ pressure-flux estimate, which it also accepted – and the mapping between every main-lane symbol of the paper's argument and its Lean counterpart, pinned to file and commit. The spine of the paper is a table of sixteen rows, each row machine-checked against the fork by the repository's test harness: the Lean declaration must exist at the cited line and at the cited commit. What the kernel can and cannot decide – including why a green verdict verifies the code but not the page – is stated as precisely as the subject allows.

1 Introduction

A green verification verdict on Lean code is a specific claim, and it is worth stating exactly what it is before using it. The Lean kernel is a type checker: it decides whether a proof term inhabits a proposition, under a fixed set of axioms. "The proof is verified" means:

1. the *statements*, as written in Lean, are well-typed propositions;
2. the *proof terms*, as written in Lean, inhabit those propositions;
3. the only axioms used are the ones on the permitted list (`propext`, `Classical.choice`, `Quot.sound` – the classical-logic baseline that mathlib itself sits on).

It does not mean that the Lean statements match the natural-language argument on the page. That is an *alignment* question, and it has its own evidence – a symbol-by-symbol mapping, and where the mapping fails, a kernel decision about the paper's actual argument. This paper is the documentation of the verified object and the alignment work, for the OpenAI fork (`NavierStokesAndEuler`, fork commit `24b1944`):

- §2: the formal statements – what the kernel actually proves, and why the statement level is faithful;
- §3: the formalization of the paper's (8.19) estimate – the kernel accepted the paper's argument as written;
- §4: the mapping table – sixteen rows, machine-checked;
- §5: the repaired $(10.19)'$ estimate – formalized, kernel-verified;
- §6: the verification – the pipeline, the signed report, the job;
- §7: limitations – what the kernel cannot decide.

The companion papers: the counterexample to (10.19) itself is [**?**]; the step-by-step walkthrough of the argument (paper, not code) is [2]; the technical walkthrough of the Lean formalization, including the parts we do not reproduce here, is [3].

2 The formal statements

The verification target is the fork's public theorem surface. The two Navier–Stokes theorems the independent reference checker (Comparator) re-checks are, at the fork's pinned state:

Lean — NavierStokes/ComparatorSolution.lean:16

```
theorem navier_stokes_breakdown_R3 (nu :  ) (hnu : nu > 0) :
      ( u :  ³ → ³) (f :  ³ →  → ³),
    InitialVelocityConditionDecay u  ForceConditionDecay f
    ¬ ( v p, NavierStokesExistenceAndSmoothnessRn nu  u f v p) := by
  exact ComparatorBridge.navier_stokes_breakdown_R3 nu hnu
```

and its periodic-variant twin, `NavierStokes/ComparatorTheorem.lean:47` (`navier_stokes_breakdown_periodic`, same shape with `...Periodic` conditions). The Euler targets: `Euler/Solution.lean:33` (`euler_breakdown_R3`) and `Euler/EulerSingularity.lean:133` (`exists_compact_smooth_euler_singularity`).

Read plainly, the Navier–Stokes statement is the paper's Theorem 1.1: for every $\nu > 0$ there are initial data and a force such that the standard existence-and-smoothness property fails – i.e. a smooth solution with the same data cannot exist, once bounded energy is assumed. The alignment audit found the statement level faithful: the Lean propositions and the natural-language theorem agree in quantifier order, hypotheses, and conclusion. The divergences found by the audit live *inside the argument* – in the pressure-flux estimate the page states as B_R-only (Lean: A_R-dependent; §4) and in the torus-inverse loss (page: $m{+}4$; code: $m{+}5$; §3). The kernel verdicts below are about the Lean object, exactly as stated here.

3 Formalizing the paper's (8.19): the kernel says the page is right

The paper's (8.19) bounds r torus derivatives of the inverse N^{-1} by $r{+}4$ input derivatives, via the divisor estimate $|v_t \cdot k|^{-1} \leq C(1{+}|k|)$. The fork's *code* proves a stronger bound – $m{+}5$ – via a product majorant with a weight^{-4} lattice sum:

Lean — NavierStokes/SmoothFamilyTorusInverse.lean:1081 (the code's own m+5 route)

```
theorem norm_derivativeWord_inverse_le (d : Direction) {f : Source P}
    (hf : ContDiff  ω f) (hp : Periodic f) (w : List Bool) (p : P) (Y : Plane) {C :  }
    (hzero : ... ‖f (p, (x, y))‖  C)
    (hfirst : ... ‖xJet (w.length + 5) (slice f p) (x, y)‖  C)
    (hsecond : ... ‖xJet (w.length + 5) (swapFunction (slice f p)) (x, y)‖  C) :
    ‖derivativeWord w (slice (inverse d f) p) ‖Y
      mixedLossConstant w.length * C := by
  ...  -- product majorant, (weight k)^-4 lattice sum
```

The question the alignment audit raised: does the *page's* $m{+}4$ argument hold? We formalized it (fork commit `9bbfa7b`), and the kernel accepted every step. The argument is: four more derivatives than requested leave a summable $(1{+}|k|)^{-3}$ majorant in two dimensions, via the symmetric split. The four lemmas, from the fork:

Lean — NavierStokes/SmoothFourierData.lean:361 (the $(1{+}|n|)^{-3/2}$ summability)

```
theorem summable_integer_weight_rpow_three_halves :
    Summable (fun n :  => (1 + |(n :  )|) ^ (-3 / 2 :  )) := by
  have hNat : Summable (fun n :  => 1 / ((n :  ) + 1) ^ (3 / 2 :  )) :=
    (Summable.congr
      ((Real.summable_one_div_nat_add_rpow (a := 1) (s := 3 / 2)).mpr (by norm_num))
      (fun n => by rw [abs_of_nonneg (by positivity)]))
  ...  -- Z = two shifted copies of N, symmetric
```

Lean — NavierStokes/SmoothFourierData.lean:381 (the symmetric split)

```
theorem weight_inv_three_le_products (k : Frequency) :
    (weight k ^ 3) ¹
      (1 + |(k.1 :  )|) ^ (-3 / 2 :  ) * (1 + |(k.2 :  )|) ^ (-3 / 2 :  ) := by
  let a := |(k.1 :  )|
  let b := |(k.2 :  )|
  ...  -- (1+a)(1+b)  (1+a+b)², hence (1+a+b)^-3  (1+a)^-3/2 (1+b)^-3/2
```

Lean — NavierStokes/SmoothFourierData.lean:413 (lattice summability)

```
theorem summable_weight_inv_three : Summable (fun k : Frequency => (weight k ^ 3) ¹) := by
  have hprod := summable_integer_weight_rpow_three_halves.mul_of_nonneg
    summable_integer_weight_rpow_three_halves (fun n => by positivity) (fun n => by positivity)
  exact Summable.of_nonneg_of_le (fun k => inv_nonneg.mpr (pow_nonneg (weight_pos k).le _))
    weight_inv_three_le_products hprod
```

The estimate itself, one derivative cheaper than the code's route:

Lean — NavierStokes/SmoothFamilyTorusInverse.lean:1109 ((8.19) as written)

```
theorem norm_derivativeWord_inverse_le_m4 (d : Direction) {f : Source P}
    (hf : ContDiff  ω f) (hp : Periodic f) (w : List Bool) (p : P) (Y : Plane) {C :  }
    (hzero :  x  Icc (0 :  ) 1,  y  Icc (0 :  ) 1, ‖f (p, (x, y))‖  C)
    (hfirst :  x  Icc (0 :  ) 1,  y  Icc (0 :  ) 1,
      ‖xJet (w.length + 4) (slice f p) (x, y)‖  C)
    (hsecond :  x  Icc (0 :  ) 1,  y  Icc (0 :  ) 1,
      ‖xJet (w.length + 4) (swapFunction (slice f p)) (x, y)‖  C) :
    ‖derivativeWord w (slice (inverse d f) p) ‖Y
      mixedLossConstant_m4 w.length * C := by
  have hbound := SmoothFourierData.coefficient_seminorm_bound_three
    (slice_smooth hf p) (hp p) (w.length + 1) hfirst' hsecond'
  calc
    _  ((6 * ‖ ‖omega¹) * ‖‖omega ^ w.length) * coeffSeminorm (w.length + 1) (coefficient f p)
      := inverse_derivativeWord_bound d (rapid_coefficient ... ) w Y
```

```
    _   ((6 * ‖ ‖omega¹) * ‖‖omega ^ w.length) *
        ((3 ^ ((w.length + 1) + 3) * C) *  ' k : Frequency, (weight k ^ 3) ¹)
      := mul_le_mul_of_nonneg_left hbound (by positivity)
    _ = _ := by simp only [mixedLossConstant_m4, Nat.add_assoc]; ring
```

(The code boxes above abridge hypothesis lists and two proof lines marked `...`; the full declarations are at the cited lines. The abridgment is flagged rather than silent because a code listing in a paper is itself a claim.)

Two readings. First, the page's (8.19) is **sound as written**: the kernel decided it, not us. Second, the divergence between page (m+4) and code (m+5) is *benign* – the code proves strictly more. The same cannot be said of the flux estimate, where the page proves strictly less and, as [**?**] shows, less than the truth.

4 The mapping table

The table below is the spine of this paper: every main-lane symbol of the paper's argument, mapped to its Lean counterpart. It is *generated* from `proof-map.tsv` (the file in the repository) – the test harness checks, for every row, that the Lean declaration exists in the cited file at the cited line *and* at the cited commit, and that the status is one of `kernel-verified` / `comparator-checked` / `repaired` / `open` / `not-checked`. A row that does not survive that check cannot appear in the rendered table.

Step	Lean declaration	Location	Commit	Status
T1.1-R3	`navier_stokes_breakdown_R3`	NavierStokes/ComparatorSolution.lean:16-16	f9e8bc5	comparator-checked
T1.1-per	`navier_stokes_breakdown_periodic`	NavierStokes/ComparatorTheorem.lean:47-47	f9e8bc5	comparator-checked
E1.1-R3	`euler_breakdown_R3`	Euler/Solution.lean:33-33	f9e8bc5	comparator-checked
E1.1-sg	`exists_compact_smooth_euler_singularity`	Euler/EulerSingularity.lean:133-133	f9e8bc5	comparator-checked
8.19-sum	`summable_integer_weight_rpow_three_halves`	NavierStokes/SmoothFourierData.lean:361-361	9bbfa7b	kernel-verified
8.19-split	`weight_inv_three_le_products`	NavierStokes/SmoothFourierData.lean:381-381	9bbfa7b	kernel-verified
8.19-summ	`summable_weight_inv_three`	NavierStokes/SmoothFourierData.lean:413-413	9bbfa7b	kernel-verified
8.19-sem	`coefficient_seminorm_bound_three`	NavierStokes/SmoothFourierData.lean:461-461	9bbfa7b	kernel-verified
8.19	`norm_derivativeWord_inverse_le_m4`	NavierStokes/SmoothFamilyTorusInverse.lean:1109-1109	9bbfa7b	kernel-verified
8.19-m5	`norm_derivativeWord_inverse_le`	NavierStokes/SmoothFamilyTorusInverse.lean:1081-1081	f9e8bc5	kernel-verified
10.3-flux	`exists_uniform_actual_pressure_flux_bound`	NavierStokes/R3/PressureFlux.lean:584-584	f9e8bc5	kernel-verified
10.19-prime	`exists_uniform_flux_bound_corrected_radii`	NavierStokes/R3/Paper1019Repaired.lean:187-187	24b1944	repaired
10.17-prime	`covering_sobolev`	NavierStokes/R3/Paper1019Repaired.lean:228-228	24b1944	repaired
10.5-a	`classical_uniqueness_on_Icc`	NavierStokes/R3/WholeSpaceUniqueness.lean:30-30	f9e8bc5	kernel-verified
10.5-b	`candidate_unique_on_Icc`	NavierStokes/R3/WholeSpaceUniqueness.lean:72-72	f9e8bc5	kernel-verified
10.5-c	`candidate_global_agrees_before_one`	NavierStokes/R3/WholeSpaceUniqueness.lean:104-104	f9e8bc5	kernel-verified

Reading the table:

- `comparator-checked`: re-verified by two independent Lean kernels (the default kernel and nanoda) under the landrun sandbox, by the Comparator reference check (§6);
- `kernel-verified`: the Lean kernel accepted the declaration in the build (every olean in the prebuilt `.lake` is re-verified against its source at build time);
- `repaired`: our formalization of the corrected estimate – kernel-verified, but flagged because it is *not* the page's statement (§5).

Two rows of the paper's argument have *no* Lean counterpart, and their absence is the point:

1. **The page's (10.19) as written** – the B_R-only flux bound. No such declaration exists in the fork; the Lean code proves the A_R-dependent bound (row `10.3-flux`). The corrected-radii version we formalized (row `10.19-prime`) is the row the page's argument *should* have.
2. **The closure of the difference-energy chain with (10.19)$'$** – the Gronwall/$R \to \infty$ limit that Lemma 10.5 consumes. Status: open. The ingredients (rows `10.5-a/b/c`, `10.19-prime`, `10.17-prime`) are all kernel-verified; the assembly is a separate formalization step (§5).

5 The repaired estimates, formalized

The minimal repair of (10.19) is a radius rescaling so that the L^6 norm is measured on a ball that *covers* the flux annulus (the paper's hierarchy B_R on B_4cR vs. the flux on $[8R, 8cR]$ is the defect; [**?**], §2). Both corrected estimates are formalized in the fork (`24b1944`, `NavierStokes/R3/Paper1019Repaired.lean`) and kernel-verified:

Lean — Paper1019Repaired.lean:187 ((10.19)' — corrected radii)

```
theorem exists_uniform_flux_bound_corrected_radii {T :  } (u v : VelocityField) (p q :
    PressureField)
    (H : PressureRecovery.Hypotheses T u v p q)
    ( M  U  G :  ) ( hM : 0   M) ( hU : 0   U) ( hG : 0   G)
    (hM :   t   Icc 0 T, MemLp (fun x => (u - v) (t, x)) 2 volume
      comparisonLpNorm 2 (fun x => (u - v) (t, x))   M)
    (hU :   t   Icc 0 T, MemLp (fun x => u (t, x)) 3 volume
      comparisonLpNorm 3 (fun x => u (t, x))   U)
    (hG :   t   Icc 0 T,   i j : Fin 3,
      Integrable (tensorDiff u v t i j) volume   comparisonLpNorm 1 (tensorDiff u v t i j)   G) :
     CP :  , 0   CP    R :  , 1   R →   t   Ioo 0 T,
      | x, (p - q) (t, x) * fderiv   (weight R) x ((u - v) (t, x))|   CP *
        ((coveringL6 R (u - v) t ^ (1 / 2 :  ) + 1) *
          (coveringDiss R (u - v) t / R + 1 / R ^ 2) + ...)
```

Here `coveringL6 R w t` is $\|w\|_6$, cutoff at $2R$ and `coveringDiss R w t` is $\left(\int \chi_2R|\nabla w|^2\right)^{1/2}$: the L^6 norm and the dissipation are measured on the ball that contains the flux annulus – the same type of bound as the fork's own `exists_uniform_actual_pressure_flux_bound` (row `10.3-flux`), with the radii corrected. The proof reuses the fork's pressure-recovery machinery verbatim; what changes is only which cutoffs the norms are taken in.

Lean — Paper1019Repaired.lean:228 ((10.17)' — covering Sobolev)

```
theorem covering_sobolev (R :  ) (hR : 0 < R) (w : VelocityField) (t :  )
    (hw : ContDiff   ω (fun x => w (t, x)))
    (hw2 : MemLp (fun x => w (t, x)) 2 volume) :
    coveringL6 R w t   weightedSobolevConstant *
      (coveringDiss R w t + derivativeConstant 1 / (2 * R) *
        comparisonLpNorm 2 (fun x => w (t, x))) := by
  have h2R : 0 < 2 * R := mul_pos (by norm_num) hR
  simpa only [coveringL6, coveringDiss] using
    WeightedSobolev.cutoffL6_le (L := derivativeConstant 1 / (2 * R))
      ((cutoff_smooth (2 * R)).of_le (by norm_num))
      (cutoff_hasCompactSupport h2R)
      (hw.of_le (by norm_num))
      hw2
```

Status, stated honestly. Both estimates are kernel-verified. What is *not* yet done: the closure of the full difference-energy inequality with $(10.19)'$ substituted, and the Gronwall/$R \to \infty$ chain that Lemma 10.5 consumes. That is the open row of the mapping table.

6 The verification

The green verdict is job `31974058-a665-472d-b6f8-00e4ec9e2d45` on the FalseGreen production stack (real job image, isolated container; verdict `accepted`; single attempt; 31.4minutes wall; signed report, schema 3). Six contract criteria, all passed:

Criterion	What the kernel / pipeline did
`build`	`lake build` in the isolated container; the Lean kernel re-verifies every prebuilt olean against its source (olean replay)
`repaired_estimate`	`Paper1019Repaired.lean` elaborated from source
`comparator_ns`	both Navier–Stokes theorems re-checked by the default Lean kernel and by the independent nanoda kernel, under the landrun sandbox
`comparator_euler`	both Euler theorems, same check
`result_audit_ns`	zero `sorry` in `NavierStokes/`; `#print axioms` of the main theorem within the permitted baseline
`result_audit_euler`	the same for the Euler full theorem

The independence layers, in order:

1. **The build.** The job workspace carries the fork's prebuilt `.lake`; the Lean kernel does not *trust* it – every olean is re-verified against its source in the checking environment, so a corrupted or doctored artifact cannot pass. (The dependency checkouts under `.lake/packages/` must carry their `.git` metadata, or the offline `lake build` deletes and re-clones them; the publish step therefore preserves exactly that metadata and nothing else of the repository's git state.)
2. **The reference check.** The Comparator re-checks the target theorems with two independent kernels – the default Lean kernel and nanoda – inside the landrun sandbox, with the prebuilt `.lake` as the sanctioned "Solution.lean will not be rebuilt" path. The challenge JSONs pin the theorems and the permitted-axiom baseline; any axiom outside the baseline fails the check.
3. **The audit.** A source-level scan counts `sorry` and prints the axioms the main theorems actually use.

The signed report is self-contained (schema 3): it embeds the frozen `falsegreen.toml` contract – verified against the job's bound `contract_sha256` and `criteria_sha256` before embedding – and, per evidence row, the typed `JobResult` (per-criterion pass/fail, per-command argv/exit/duration). The `evidence_sha256` commitments can be re-derived from the embedded rows, and the report signature – an HMAC-SHA256 over the canonical report digest, keyed with the deployment's report key – can be re-verified offline (`falsegreen-canary verify-report`) without contacting the platform. Appendix A summarizes the report's structure.

7 Limitations: what the kernel cannot decide

1. **The kernel does not check the page.** It decides whether the Lean object proves itself. The alignment question – do the Lean statements and the page's argument mean the same thing? – is decided by reading, with the kernel as referee for the parts that *are* formalizable: (8.19) as written (§3), and the repaired $(10.19)'$ (§5). The part that is not formalizable – the support bookkeeping of the page's (10.19) – is decided by a counterexample, not by the kernel [**?**].
2. **A green verdict is not a theorem endorsement.** It endorses the Lean object. Whether that object is the paper's argument is the mapping table of §4: faithful at the statement level, divergent at two argument-level points, one benign ((8.19)), one not ((10.19)).
3. **The open row.** The closure of the difference-energy chain with $(10.19)'$ is not yet formalized. Until it is, "the repaired argument works" is a reading, not a verdict – the honest status of the repair, and the next formalization step.

8 Related work

The verification tooling is summarized in the job's signed report (Appendix A) and in the Lean ecosystem's own documentation of the Comparator's trust model [5] and of nanoda's independent kernel [4]. For the alignment methodology – paper ↔ code, with the kernel as referee for the formalizable parts and a counterexample for the rest – the four papers of this series are the worked instance: [**?**] (the counterexample), [2] (the step-by-step walkthrough of the argument), [3] (the technical walkthrough of the formalization). Bastounis, Circelli, and Hansen [1] document the same "lost in translation" phenomenon from the autoformalization side; this paper is the verification-side account of the same gap.

References

[1] Hansen et al. Navier–stokes lost in translation. arXiv:2610.08144, 2026. Semantic faithfulness of AI autoformalisation; worked examples on the OpenAI NS proof.

[2] Bevan Hunt. The navier–stokes proof, step by step: What OpenAI claimed, what the machines checked, and where the argument breaks. Companion paper, this series, 2026. DOI: PENDING.

[3] Bevan Hunt. Inside the verified proof: A step-by-step walkthrough of the Lean navier–stokes formalization. Companion paper, this series, 2026. DOI: PENDING.

[4] Sander R. J. J. Janssen et al. nanoda: a standalone Lean kernel. URL https://github.com/ammkrn/nanoda.

[5] OpenAI. Comparator: independent reference checking for Lean proofs (landrun + nanoda). Dual-kernel reference checks: default Lean kernel + independent nanoda kernel, under the landrun sandbox.

A The verification report, in structure

Report schema 3 (job `31974058-a665-472d-b6f8-00e4ec9e2d45`):

- **Verdict block.** `accepted`; job id; record sequence; signature (HMAC-SHA256 over the canonical report digest, keyed with the deployment's report key – re-verifiable offline via `falsegreen-canary verify-report`); attempt accounting (1 of 8 budgeted attempts used).
- **Contract.** The frozen `falsegreen.toml` – the "things to prove" (criteria) and the "how" (commands) – embedded only after verification against the job's bound `contract_sha256/criteria_sha256`; a contract that cannot be proven to be the bound one is omitted (fail-closed).
- **Evidence.** One typed `JobResult` per criterion: per-criterion pass/fail, per-command argv/exit/duration/output-tail, environment capture (the verifier's identity hash must equal the protocol's canonical hash – a regression test locks this).
- **Commitments.** `evidence_sha256` per row (re-derivable from the embedded rows) and the overall report signature.

A later reassessment is a new record, not an edit: the reader serves the latest record sequence; the first record is immutable.