hexagon:2611.01313v1

LEANXRAY, ZEROCOMMITSLICE, AND BOUNDED-MEMORY VERIFICATION: DOMINATOR AUDITING, ZERO-COMMIT COMPILATION, AND STREAMING PROOF-BODY ANALYSIS FOR MILLION-DECLARATION LEAN 4 PROOFS

ROBERT KADLEC

Abstract. Large-scale AI-assisted Lean 4 formalizations now exceed 1.26×10^6 declarations and 94 GiB of compiled `.olean` archives, while single-module `lean4export 3.1.0` streams reach 445 MiB and commit ≈ 1 GiB of OS memory in existing external checkers. We present `lean-slice-oss`, an Apache-2.0 engineering suite enabling million-declaration auditing and verification on 32 GiB workstations via four tools: (1) **LeanXRay**, combining Cooper–Harvey–Kennedy dominator auditing with native `.olean` structure inspection; (2) **ZeroCommitSlice**, a portable Windows/POSIX zero-commit `.olean` loader and reverse-topological slicer; (3) `lean-export-stream`, a Rust streaming structural/axiom auditor and proof-body liveness modeler; and (4) a 3-patch series for `nanoda_lib 0.4.32` introducing a file-backed `mmap` expression arena and tagged axiom reporting. Across three real exports, the disk arena cuts peak Windows pagefile commit by 4.16×–14.64× and Linux `RssAnon` by 4.10×–8.51×. A cross-OS causal ablation shows that disabling theorem-body unfolding breaks Mathlib typechecking on `Rat.instEncodable`, proving permanent proof-body deletion incomplete and motivating reclaimable `mmap` arenas.

1. Motivation: Five Engineering Bottlenecks at Million-Declaration Scale

When auditing a 1.265×10^6-declaration formal proof [1] or cross-verifying multi-hundred-megabyte `lean4export` streams [6] on a commodity 32 GiB workstation, standard interactive workflows hit five engineering bottlenecks:

(1) *Monolithic coupling and regex fragility:* In the Alpöge archive [1] (historical companion-archive analysis [5]), deleting all 194 literature leaf theorems in `PrincipalDirect.lean` disconnects only 244,592 declarations (19.34% of the 1,264,848-node DAG); $> 80.6\%$ of the graph lives below five structural chokepoints. Source regexes misclassify local `haveI` declarations as structure fields and miss compiled kernel `Expr.proj` nodes.

(2) *Pagefile commit exhaustion and `MAX_PATH`:* Mapping $> 100{,}000$ `.olean` files with default copy-on-write mappings exhausts OS commit limits on 32 GiB hosts, while deep module hierarchies exceed Windows' 260-character path limit.

(3) *Random-order BFS disk thrashing:* Queue-based reachability sweeps over 175,173 memory-mapped `.olean` files jump randomly across 94.3 GiB of disk pages, stalling below 100 declarations/second.

(4) *Declaration-DAG vs. full-import closure gap:* Computing reverse-import closures solely inside the 146,398-module declaration DAG omits zero-declaration or off-cone bridge modules sitting on transitive import paths between modified modules (encountered live with `CarFillE` and `WindowBoundOfLiteratureLevel` below `TopMainZj`).

(5) *In-RAM export commit pressure and theorem-unfolding completeness:* External Rust checkers holding the entire expression arena in RAM commit ≈ 1 GiB (1,020 MiB) of OS pagefile charge

Date: November 28, 2026.

ORCID: 0009-0003-9245-8130. Purely AI-generated research directed and supervised by the author, by whom full scholarly responsibility for the work is assumed (all new implementation, refactoring, scripts, cross-model audits, and prose were executed by AI systems; see Section 5). This paper is part of a broader research project in which the respective competencies of humans and large language models are investigated.

for a single 445 MiB export, yet permanently discarding theorem proof bodies after checking risks false rejections (*incompleteness*) whenever the kernel unfolds a theorem body during definitional equality checking (e.g., via `Acc.rec` or subsingleton elimination).

2. LeanXRay and Source-Level Cross-Archive Auditing (`lean_dag_audit.py`)

2.1. **Binary DAG Cache, Dominator Trees, and CSR Extraction (Historical Companion Receipts).** In our companion Paper 2 release archive [5] (historical receipts, not regenerated inside this repository's `receipts/` directory), our Python/NumPy pipeline caches the 1,264,848-node, 14,075,546-edge Alpöge DAG in 76.5 MiB (`dag_cache.npz`) and computes exact immediate dominator trees [7, 4] in ~ 7 s. Because edge-cut savings are non-additive across overlapping cones, joint multi-cut BFS on the cached graph simulated $-3{,}751$ declarations for $P + F + \mathrm{cm} + \mathrm{Hironaka}(1, 2)$ (19 surface landmarks) and $-4{,}037$ net declarations when stacking Stickelberger and `UnusedSlot33` (compared with measured live-cut deltas in Table 2, which track full kernel-reachable closures rather than declaration-DAG-only simulations). Similarly, the CSR Mathlib candidate scanner `scan_mathlib_candidates.py` was originally timed at 0.84 s over the cached 1.265M-node graph in [5] (isolating `GroundTheory.TateCohomology`, `GroundTheory.FunctionField`, 14 zero-external-dep subgraphs, and Hironaka's single 806-byte external dependency `SaturationByPowers`); in `lean-slice-oss`, we harden it with CSR bounds/monotonicity validation and unit tests on synthetic CSR fixtures.

2.2. **Native `.olean` Projection, Eliminator, and Constructor Auditing (`LeanXRayDump.lean`).** Our native Lean 4.33.1 metaprogram `LeanXRayDump.lean` queries `Lean.getStructureInfo?` on compiled `.olean` environments, walks every reachable expression DAG with a memoized `Std.HashSet Expr` (avoiding exponential re-traversal of shared subexpressions and covering recursor rule bodies), and exits non-zero if any referenced constant is missing. Compile-time selftests exposed four non-obvious kernel requirements [9]:

Proposition 2.1 (Four Syntactic Requirements for Lean 4 Structure Auditing)**.** *To accurately identify fields of a structure S with no syntactic projection references in the dependency cone of a root declaration R under Lean 4.33.1:*

1. Theorem and opaque unpacking: *`Lean.ConstantInfo.value?` (with its default `allowOpaque := false`) returns `none` on both `.thmInfo` and `.opaqueInfo`; the walker must explicitly unpack `.thmInfo v => v.value`, `.defnInfo v => v.value`, `.opaqueInfo v => v.value`, and `.recInfo v => v.rules.map (·.rhs)`.*
2. Both `Expr.proj` and named projections: *Lean 4 compiles field accesses to both primitive `Expr.proj S idx e` and named constants `Expr.const S.field`; walking only `getUsedConstantsAsSet` misses all `Expr.proj` uses.*
3. Field index order: *Inside `Lean.StructureInfo`, `info.fieldInfo` is sorted by `Name.quickLt` for binary search, whereas kernel `Expr.proj` indices $0, 1, \ldots$ follow declaration order `info.fieldNames`.*
4. Eliminator and constructor hazards (advisory scope): *The flag `advisoryNoProjOrListedElimInCone` requires zero listed eliminators (`rec`, `casesOn`, `recOn`, `ndrec`, `ndrecOn`, `below`, `brecOn`, `noConfusion`) and zero constructor uses (`S.mk`) in the cone. Even when `true`, it remains strictly* advisory *and requires recompilation and kernel checking of the affected transitive closure.*

All three compile-time selftests (`demo_consumer_proj`, `demo_consumer_elim`, `demo_constructor_only`) execute during `lean -c` compilation and pass automatically.

2.3. **Source-Only Cross-Archive Auditing (`lean_dag_audit.py`).** We generalized module-level Cooper–Harvey–Kennedy [4] dominator trees, extraction censuses, and lexer-aware forbidden-token

scanning into `lean_dag_audit.py`, requiring *no* precompiled `.olean` files. Table 1 summarizes receipts regenerated directly by the shipped CLI across Chudnovsky's transcendence archive [2] and Sernau's compactified GRH repository [11].

TABLE 1. Source-only cross-archive audit via the shipped `lean_dag_audit.py` (`receipts/o04_cross_archive_audit.json`). Declaration counts are source-level regex counts (`decls_regex`). Every scanned archive recorded 0 forbidden tokens under `-root` (`sorry`/`sorryAx`, `admit`, `native_decide`, `decide +native`, `bv_decide`, `axiom`, `unsafe`, `implemented_by`, `extern`, `ofReduceBool`, `trustCompiler`); this is a source hygiene scan, not `#print axioms`.

External Lean 4 Archive	Modules	Lines	Regex Decls	Local Edges	Target Cone	Unused Mods
Chudnovsky `2610.26539` [2] (full)	976	114,709	8,191	1,997	969 mods (8,169 decls)	7
`BlowupJetSurjectivityComplete` subcone	—	—	—	—	225 mods (2,203 decls)	751
`lsernau/grh-lean` [11] (`grh-compact/`)	794	112,827	10,946	1,398	794/794 (100%)	0
`lsernau/grh-lean` [11] (`grh/`)	803	477,124	23,839	1,733	803/803 (100%)	0

In particular, `lean_dag_audit.py` verified that Chudnovsky's 828-module `OAI.NumberTheory.PiExponent` tree has 0 imports from `QuadraticPi` and its ampleness subcone (`BlowupJetSurjectivityComplete`) spans 225 modules (28,423 lines), while `lsernau/grh-lean` has 0 unused modules outside the cone of `Main`.

3. ZeroCommitSlice: Portable Zero-Commit Loading and Live Dependency Cuts

3.1. Cross-Platform Zero-Commit Loader and Reverse-Topological Sweep. `ReadonlyArchive_portable.c` is pinned to Lean 4.33.1 (commit `819816b2`, 64-bit little-endian GMP encoding checked via `_Static_assert` and build-time sysroot validation), exposes the exact 2-argument C ABI emitted by Lean 4.33.1 for `alpoge_read_module` (verified at compile time by `build.py` via `-include generated_abi_check.h` under `-Wall -Wextra -Werror`), and replaces copy-on-write `.olean` mapping on archive roots with:

- **Windows:** `CreateFileMappingW(..., PAGE_READONLY)` + `MapViewOfFileEx` at the saved `.olean` base address (zero OS pagefile commit charge).
- **Linux / POSIX:** canonicalized `realpath` containment + `S_ISREG` validation followed by `mmap((void*)base, len, PROT_READ, MAP_PRIVATE | MAP_NORESERVE | MAP_FIXED_NOREPLACE, fd, 0)` (with optional `MAP_NORESERVE` and sanitizer guards, plus fallback to stock 3-arg `lean_compacted_region_read(..., lean_io_mk_world())`). On Ubuntu 26.04.1 `x86_64` (kernel `7.0.0-38-generic`, Lean 4.33.1), both `gcc 15.2.0` and Ubuntu `clang 21.1.8` compile `ReadonlyArchive_portable.c` with `-Wall -Wextra -Werror` and the generated 2-arg ABI header with 0 warnings, and 5/5 live POSIX runtime checks pass (`ZeroCommitSlice [POSIX]: mapped=1 bytes=6848 fallbacks=0`, separate `reused_existing=1` manifest accounting, `LeanXRayDump` live `.olean` inspection, positive-control symlink-escape fallback rejecting an external symlink while mapping an in-archive import, and RUNPATH startup without `LD_LIBRARY_PATH`; `receipts/linux_20261128/linux_posix_mmap_runtime.json`). Clean-machine Linux QA exposed four build-harness defects—Linux `leanc -leanshared` omitting `-lleanshared` (D2), `build.py` prepending `<sysroot>/bin` so `-cc clang` shadowed system `clang` with Lean's headerless bundled `clang` (D3), `Makefile` omitting `#include <lean/lean.h>` from `generated_abi_check.h` (D4), and missing `-Wl,-rpath,<sysroot>/lib/lean` RUNPATH (D5)—all repaired in `c_lean/build.py` and `c_lean/Makefile`.

Linked with `SliceFrontendV2_OSS.lean` (which checks circular imports via active-DFS tracking, validates nonempty targets, walks `Expr.proj` struct names and recursor rule bodies, tracks `compiled`

vs. `reused_existing` outputs separately, and writes JSON receipts via `-receipt`), our reverse-topological sweep visits imported modules $b = N-1, N-2, \ldots, 0$ (touching each `.olean` file at most once). In the historical Alpöge RH runs [5], this mapped 174,155–175,042 standalone archive files (93.6–94.3 GiB, 118–124 fallbacks) on a 32 GiB host with peak working set 11–17 GiB, sweeping $\sim 1.473 \times 10^6$ constants in 653.0–799.3 s (1,843–2,256 decls/s).

3.2. **Historical Live End-to-End Dependency Cuts (Companion Paper 2 Archive).** Table 2 records the three historical live-compiled dependency cuts from our companion Paper 2 release [5] through `PrincipalDirect`, `PrincipalSupply`, and `CutClosedRH` (`closedRH : RiemannHypothesis`), where expanding the `CM` reverse-import closure over the full 175,180-node import graph (175,173 archive `.olean` modules plus 7 top-level project/entry files; 7 additional non-imported helper files bring total `.lean` files to 175,187) caught two non-DAG bridge modules (`CarFillE`, `WindowBoundOfLiteratureLevel`; 400 → 402, total 897 archive modules, 0 topological violations).

TABLE 2. Historical live kernel-BFS cones (`SliceFrontendV2`, Lean 4.33.1, companion archive [5]) across three progressive RH cuts. Every target passed `Environment.addDeclCore` with axioms strictly `[propext, Classical.choice, Quot.sound]` relative to trusted read-only archive imports. Stage 2 tracks 21 landmarks (19 surface + 2 sub-landmarks); Stage 3 adds 6 Stickelberger landmarks (27 total).

Live Cut Stage	Arch. Mods	`hSup` Total	`closedRH` Total	Arch+Proj	Arch+Proj Mods	Landmarks Absent
Stage 1: P + Hironaka$_{1,2}$	7	1,477,330	1,477,360	1,263,979	146,187	3/21
Stage 2: $P + F$ + CM + Hironaka$_{1,2}$	897	1,473,424	1,473,454	1,260,117	145,643	21/21
Stage 3: Stage 2 + Stickelberger (C3)	897 + 0	1,473,199	1,473,229	1,259,895	145,601	27/27
Measured Delta (Stage 3 vs. Stage 1)	—	−4,131	−4,131	−4,084	−586	27/27 **unreached**

4. Streaming Export Auditing, Proof-Body Liveness, and `NANODA_LIB` Patches

4.1. **Single-Pass Streaming Auditing and Structural Liveness Model (`lean-export-stream`).** To pre-flight and analyze multi-hundred-megabyte `lean4export 3.1.0` NDJSON streams without constructing a full expression DOM in single-pass mode, we ship both a Python auditor (`stream_export_audit.py`, 6.738 MiB peak traced Python heap via `tracemalloc` on a 34.38 MiB export, 12/12 reason-checked synthetic tampers rejected in `receipts/o02_python_benchmark_receipt.json`) and a compiled Rust tool `lean-export-stream` (compiled with `-D warnings`). Both implement strict duplicate-key JSON rejection, require exactly one initial `meta` record with format `3.1.x`, reject unknown/multi-key records, enforce contiguous IDs and back-references across every record kind (`in`, `il`, `ie`, `axiom`, `def`, `thm`, `opaque`, `inductive`, `quot`, `levelParams`, `all`), compute SHA-256 digests, and compare permitted axioms as exact tagged `Name` component sequences (defeating single-component `«Quot.sound»` collisions). In `cargo test`, `lean-export-stream` runs 2 Rust unit tests (`cargo test` reports 2 passed), verifying an exporter-valid positive fixture and 32 diagnostic assertions rejecting 16/16 tampered stream classes across both single-pass and `-liveness` modes.

Its $O(\#\text{exprs})$ `-liveness` mode partitions exported expressions into those reachable from types, definitions, opaques, inductives, axioms, or quotients (`retained_non_proof_exprs`) versus those reachable *exclusively* inside theorem proof terms (`theorem_proof_only_exprs`), while also reporting how many proof-only nodes are shared across $\geq$ 2 theorem closures (`shared_proof_only_exprs_ge2_thms`). Across three Lean 4.33.1 exports generated from our companion formalization suites—`DVRConductorNoGoingUp` (Paper 4 DVR conductor descent, 34.38 MiB), `KroneckerFpCert` ($\mathbb{F}_p[x]$ kernel polynomial certification, 96.98 MiB), and `PostRHAbelianClosure`

(Paper 2 `v2.1.0-pre` abelian window criterion, 445.01 MiB)—`lean-export-stream` sustains 88.9–95.9 MiB/s and measures proof-only shares of 60.95%, 66.00%, and 74.65%, corresponding to serialized structural expression-count ratios (`modeled_serialized_expr_ratio`) of 2.46×, 2.88×, and 3.83× (Table 3). This ratio is strictly a structural node-count model ignoring execution order, byte sizes, and OS paging.

4.2. **Empirical Kernel Completeness Finding, Causal Ablation, and Upstream `nanoda_lib` 3-Patch Series.** It is tempting to infer from the 61%–75% proof-only share that a streaming Lean 4 checker can permanently delete every theorem proof body immediately after checking it. Instrumenting `nanoda_lib`'s `TypeChecker::unfold_def` with a per-`ExportFile` atomic counter incremented only after universe-arity matching succeeds on a `Declar::Theorem` first revealed 94, 258, and 1,211 theorem proof-body unfolds across our three exports. To test causality—whether those unfolds are strictly necessary for kernel completeness rather than merely alternate reduction paths—we executed a dedicated test-only ablation (`receipts/theorem_unfold_ablation_windows.json`) in which `TypeChecker::get_applied_def` and `TypeChecker::unfold_def` return `None` for `Declar::Theorem` (treating previously checked theorems as body-less opaques) and `check_all_declars_serial` catches panics per declaration:

- `DVRConductorNoGoingUp.export`: 34 blocked theorem-unfold attempts → 0/8,410 failures (`exit 0`);
- `KroneckerFpCert.export`: 136 blocked theorem-unfold attempts → 0/20,441 failures (`exit 0`);
- `PostRHAbelianClosure.export`: 473 blocked theorem-unfold attempts → 2/63,537 **hard kernel typechecking failures** (`exit 101`, `assertion failed: self.def_eq(u, v)`) on standard Mathlib declarations `Rat.instEncodable._proof_3` and `Rat.instEncodable`.

Permanently discarding theorem proof terms after checking is therefore *provably incomplete* on real Mathlib proofs.

Instead, backing the expression interner with a **file-backed `mmap` arena** removes expression storage from the OS pagefile commit charge on Windows while keeping every proof body faultable on demand. We ship a self-contained 3-patch series targeting pinned upstream `ammkrn/nanoda_lib` `0.4.32` commit `0da9f98feadd3a0a2ef4b3dbcf52a298a6fef929` (cleanly verified with `git apply -check -whitespace=error`, 58/58 `cargo test -release` passing):

(1) **Patch 1 (File-backed `mmap` `ExprStore`):** Adds `crate::disk_expr::{Disk, ExprStore}` with Windows `create_new + share_mode(0)` and POSIX `0700`/`0600` exclusive creation plus immediate post-`mmap` unlink, clean `Drop` teardown, and `try_insert_full` error propagation up to 2^{31} expressions.

(2) **Patch 2 (`W2-1` Fix + Tagged Axiom Receipts):** Removes the spurious `"Unable to print axioms"` error when `pp_to_stdout: false`, switches `PpDestination` to `write_all + flush()`, enforces component-level axiom whitelist matching (rejecting `«Quot.sound»`), and adds atomic non-aliasing `axioms_json_path`.

(3) **Patch 3 (`O-02` Single-File `disk_arena_dir` + Telemetry):** Exposes `disk_arena_dir`, per-`ExportFile` `theorem_unfolds`, and actual `is_disk_arena()` / `disk_arena_bytes()` reporting with no intended semantic change to kernel checking.

On the 445.01 MiB `PostRHAbelianClosure.export` (63,537 declarations), `disk_arena_dir` reduces Windows peak pagefile commit by 14.64× (1,020.07 → 69.70 MiB) and peak working set by 1.68× (734.19 → 436.25 MiB), running within +6.0% of patched RAM (46.98 vs. 44.32 s) and 1.3% faster than unpatched upstream (47.60 s); on DVR and Kronecker, single-run wall times change by +9.2% and +2.3% while peak working set drops 1.23× on DVR and rises +0.4% on Kronecker. All three backends (unpatched `0da9f98f`, patched RAM, and patched Disk) produce identical acceptance verdicts and declaration counts, and the 4-control paired `sorryAx` suite verifies whitelist admission and exact `unpermitted axiom "sorryAx"` rejection on both RAM and Disk.

TABLE 3. Measured `lean-export-stream` throughput/liveness and upstream `nanoda_bin` A/B benchmarks on three Lean 4.33.1 exports (`receipts/rust_stream_*.json` and `receipts/nanoda_patches_ab_benchmark.json`, single-threaded Windows, `pp_to_stdout: false`, `print_axioms: true`). Unpatched `0da9f98f` emitted 1 pretty-printer error on every run; all patched runs emitted 0 errors, 0 leftover arena files, and identical declaration/axiom/unfold counts. A 4-control paired schema-valid `sorryAx` policy suite passed on both RAM and Disk.

Export (Lean 4.33.1)	Size (MiB)	Unique Exprs	Stream (MiB/s)	Proof-Only % (Mod. Ratio / Sh.≥ 2)	Unpatched Commit / Time	RAM Commit / WS / Time	Disk Commit / WS / Time	Commit Drop	Thm. Unf.
`DVRConductorNoGoingUp`	34.38	629,021	93.4	60.95% (2.46× / 31k)	66.49 MiB / 1.37 s	66.08 / 51.29 MiB / 1.39 s	13.79 / 41.76 MiB / 1.51 s	4.79×	94
`KroneckerFpCert`	96.98	1,769,373	95.9	66.00% (2.88× / 95k)	131.88 MiB / 5.62 s	131.21 / 111.51 MiB / 6.32 s	31.52 / 111.91 MiB / 6.47 s	4.16×	258
`PostRHAbelianClosure`	445.01	8,233,903	88.9	74.65% (3.83× / 746k)	1,020.05 MiB / 47.60 s	1,020.07 / 734.19 MiB / 44.32 s	69.70 / 436.25 MiB / 46.98 s	14.64×	1,211

4.3. **Clean-Machine Linux `x86_64` Replay and `MAP_SHARED` Filesystem Accounting.** A clean-machine Linux `x86_64` replay pass (Ubuntu 26.04.1 LTS, kernel `7.0.0-38-generic`, AMD EPYC 7713P, 62 GiB RAM, no swap, `rustc 1.96.0`, isolated via `systemd-run -user` with 24 GiB memory cap; anonymized receipts in `receipts/linux_20261128/`) reproduced all semantic results across Windows and Linux: 3/3 `lean-export-stream` exports matched on every structural, axiom, and liveness field (64.9–69.9 MiB/s); 6/6 patched `nanoda` summary JSONs were byte-identical to Windows (94/258/1,211 theorem unfolds, 32/88/384 MiB disk arenas, 58/58 `cargo test`, 4/4 `sorryAx` policy controls); POSIX immediate-`unlink` kept visible arena files at 0 throughout every run with 0 leftovers and 0 major page faults; and the Stage 5 theorem-unfold ablation identically failed on the same 2/63,537 declarations (`Rat.instEncodable._proof_3` and `Rat.instEncodable`, `src/tc.rs:1036`). Table 4 reports the Linux memory and wall-time metrics.

TABLE 4. Clean-machine Linux `x86_64` A/B verification (`receipts/linux_20261128/linux_nanoda_ab_benchmark.json`, single-threaded, Ubuntu 26.04.1 / EPYC 7713P, arena on `tmpfs`, 0 major faults (7/20/81 with the arena on `ext4`), 0 visible arena files during run, 6/6 summary JSONs byte-identical to Windows).

Export	Stream (MiB/s)	Unp. / RAM / Disk Wall (s)	Unp. / RAM / Disk `ru_maxrss` (MiB)	RAM → Disk `RssAnon` (MiB)	`RssAnon` Drop	RAM → Disk `VmData` (MiB)	Vis. Files
`DVRConductorNoGoingUp`	64.9	2.10 / 2.26 / 2.50	46.6 / 46.6 / 40.0	44.8 → **10.9**	**4.10**×	90.4 → 45.6	0
`KroneckerFpCert`	65.9	8.24 / 8.98 / 9.51	112.3 / 112.8 / 106.5	111.1 → **17.7**	**6.27**×	154.0 → 54.9	0
`PostRHAbelianClosure`	69.9	60.2 / 65.2 / 71.9	532.2 / 532.3 / 443.4	526.7 → **61.9**	**8.51**×	812.6 → **102.8**	0

A supplementary filesystem experiment (`X1`, `x1_arena_tmpfs_vs_ext4.log`) on `PostRHAbelianClosure.export` uncovered an important Linux systems nuance for `MAP_SHARED` arenas: while `RssAnon` drops 8.5× in both configurations and `VmHWM` stays at 443.2 MiB, placing `disk_arena_dir` on RAM-backed `tmpfs` (`/tmp`) accounts 380.4 MiB as non-reclaimable `RssShmem` (59.9 MiB `RssAnon`, 2.1 MiB `RssFile`, 73.1 s), whereas placing it on disk-backed `ext4` (`/var/tmp`) moves 381.2 **MiB into kernel-reclaimable `RssFile` page cache** (60.5 MiB `RssAnon`, 1.1 MiB `RssShmem`, 74.7 s). A hard cgroup limit check (`X2`, `MemoryMax` = 320/160 MiB < VmHWM, no swap) further showed that RAM mode and a `tmpfs` arena suffer OOM kills within ∼ 5 s, whereas an `ext4` arena avoids OOM (`oom_kill` = 0, ∼ 140k major faults) but thrashes (∼ 92% system time, exceeding 480 s vs. 74–76 s unconstrained)—reclaimability prevents OOM kills, not paging slowdowns when RAM is far smaller than the active working set. On Linux, `disk_arena_dir` should therefore target a disk-backed local filesystem (e.g., `/var/tmp` on `ext4`/`XFS`), avoiding both `tmpfs` (non-reclaimable without swap) and NFS (where unlinked open files create `.nfs*` silly-rename entries).

4.4. **Minimal `con-leche` v4.36.0 Decline Reproducer.** Our release packages `PrincipalizationCertRepro.lean` (26 lines of Lean code; 31 lines with header; zero Mathlib), in which a reflexive mutual inductive predicate in `Prop` with function-typed recursive hypotheses over a higher universe suffices to trigger `con-leche`/`con-ron` v4.36.0's `-verified` decline (`exit 2`, `reflexive member` [3]), alongside two passing controls (`ConLecheNestedRepro.lean`, 18/27 lines; `ConLecheMutualRepro.lean`, 19/28 lines; all 5 checkers exit 0).

5. Strict Epistemic Boundaries and AI Provenance

- `lean-export-stream` and `stream_export_audit.py` perform structural, duplicate-key, back-reference, SHA-256, liveness-modeling, and tagged-component axiom-*name* checks only; they do *not* verify axiom type signatures and do *not* replace full dependent typecheckers. Single-pass auditing avoids an expression DOM; `-liveness` uses $O(\#\text{exprs})$ memory.
- `lean_dag_audit.py` performs source-level regex/lexer hygiene scanning over files under `-root` only and does not replace `#print axioms`.
- `LeanXRayDump` reports a strictly syntactic advisory flag, not a semantic proof of safe field deletion without recompilation.
- Historical Alpöge live-cut and CSR figures in Section 2.1 and Table 2 derive from companion archive [5], where `PrincipalSupply.hSup` and `closedRH` passed Lean 4.33.1's `Environment.addDeclCore` relative to trusted read-only archive `.olean` imports.
- Both Windows `x86_64` and Linux `x86_64` (Ubuntu 26.04.1, `gcc 15.2.0`, Ubuntu `clang 21.1.8`, `rustc 1.96.0`, Lean 4.33.1) have been live-tested across all Python, Rust, `nanoda_lib`, and C/Lean suites (`receipts/` and `receipts/linux_20261128/`); Linux CI jobs remain defined in `.github/workflows/ci.yml`. On Linux, actual RAM-pressure reclaimability of `disk_arena_dir` requires a disk-backed local filesystem (`ext4`/`XFS`), not `tmpfs` or NFS; macOS has not been tested locally.

Pure-AI Provenance and Human Supervision. This manuscript and the accompanying `lean-slice-oss` software suite report purely AI-generated engineering and research directed, scoped, and supervised by the human author (Robert Kadlec, Masaryk University, Brno, Czech Republic; email `kadlecr@mail.muni.cz`; ORCID 0009-0003-9245-8130), as part of a broader research project in which the respective competencies of humans and large language models are investigated. All new code, patches, benchmark harnesses, cross-model adversarial audits, and prose were executed by AI systems (Google Gemini/Antigravity, OpenAI Codex, and Anthropic Claude), while full scholarly responsibility for the integrity, accuracy, and publication of all reported results is assumed by the human author. Full audit logs and provenance records are archived in `AI_DISCLOSURE_AND_PROVENANCE.md`.

References

[1] L. Alpöge, *Lean 4 Formalization of the Riemann Hypothesis (Archive Revision `79c751e0b0a5ea3bbfc20198c852fa85f28661e6`)*, 2026.

[2] G. V. Chudnovsky, *Lemniscate irrationality exponent via separated-weights interpolation in Lean 4*, arXiv:2610.26539, October 2026.

[3] `con-leche` / `con-ron` (v4.36.0), *Third-party Lean 4 external kernel verifiers*, 2026.

[4] K. D. Cooper, T. J. Harvey, and K. Kennedy, *A simple, fast dominance algorithm*, Software Practice & Experience **4** (2001), 1–10.

[5] R. Kadlec, *Auditing the Principal-Character Cone of Alpöge's Lean Proof of the Riemann Hypothesis (Release v2.0.0)*, Zenodo, `https://doi.org/10.5281/zenodo.24584643`, November 2026.

[6] Lean FRO / `lean4export` (format v3.1.0), `https://github.com/leanprover/lean4export`, 2026.

[7] T. Lengauer and R. E. Tarjan, *A fast algorithm for finding dominators in a flowgraph*, ACM Trans. Program. Lang. Syst. **1** (1979), no. 1, 121–141.

[8] The mathlib Community, *The Lean mathematical library*, Proc. 9th ACM SIGPLAN Int. Conf. Certified Programs and Proofs (CPP 2020), pp. 367–381.

[9] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, LNCS 12699, Springer, 2021, pp. 625–635.

[10] ammkrn, *nanoda_lib: A Lean 4 type checker in Rust (v0.4.32, commit 0da9f98f)*, https://github.com/ammkrn/nanoda_lib, 2026.
[11] L. Sernau, *grh-lean: Compact Lean 4 Formalization of GRH*, https://github.com/lsernau/grh-lean, November 2026.

Masaryk University, Brno, Czech Republic
Email address: kadlecr@mail.muni.cz