Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Not Letting Proofs Die: xLaDe, a Lean 4 Ecosystem Framework

Lakshit Singh Bisht

lakshitsinghbishttm@gmail.com

Central University of Andhra Pradesh https://orcid.org/0009-0004-0035-5714

Deepak Pandey

Central University of Andhra Pradesh

Manpreet Sahoo

Central University of Andhra Pradesh

Thota Siva

Central University of Andhra Pradesh

Research Article

Keywords: Lean 4, Proof Assistants, Reproducibility, Formal Verification, Software Preservation

Posted Date: September 15th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11019636/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Not Letting Proofs Die: xLaDe, a Lean 4 Ecosystem Framework

Lakshit Singh Bisht[1*], Deepak Pandey[1], Manpreet Sahoo[1], and Thota Siva[1]

[1]Central University of Andhra Pradesh

*Corresponding author. E-mail: lakshitsinghbishttm@gmail.com

Abstract

Lean 4 is under active development, and proofs that compile under one version may fail under later versions due to changes in the language, elaborator, libraries, or tooling. While the Lean community accepts this trade-off in exchange for rapid improvement, it creates challenges for long-term reproducibility of formalized mathematics and research artifacts.

In this paper, we present xLaDe (eXperimental Lean 4 Advanced Development Ecosystem), a Lean 4 ecosystem framework designed to support reproducibility, preservation, and experimentation. xLaDe records toolchain metadata, dependencies, and execution context, enabling reconstruction of proof environments long after their original creation. The framework treats the Lean kernel as immutable infrastructure and wraps external Lean 4 projects without modifying their source code, allowing experiments to be conducted while preserving project integrity.

xLaDe provides policy enforcement, experiment management, automated verification, and decentralized distribution mechanisms for Lean 4 projects. In this paper, we describe the design, implementation, security model, and experimental evaluation of xLaDe, and discuss its role as an ecosystem layer for improving the long-term sustainability of Lean 4 development and research.

Keywords: Lean 4, Proof Assistants, Reproducibility, Formal Verification, Software Preservation

1 Introduction

Proof assistants have become increasingly important in mathematics, software verification, and formal methods. Systems such as Lean[1], Coq[2], Isabelle/HOL[3], HOL Light[4], and Agda[5] are used to construct machine-checked proofs and verify the correctness of software and mathematical results. Among these systems, Lean 4[6] has emerged as a modern theorem prover that combines dependent type theory with a practical programming language and an actively developed ecosystem. Large collaborative projects such as Mathlib[7] have demonstrated that substantial bodies of formalized mathematics can be developed and maintained within this environment.

The success of modern proof assistants depends not only on their logical foundations but also on the surrounding ecosystem. Toolchains, package managers, libraries, build systems, documentation, and workflows collectively influence how research is conducted and reproduced. As proof assistant ecosystems continue to grow, reproducibility becomes increasingly important. A proof or experiment that compiles under one toolchain version may fail under a later version due to changes in dependencies, libraries, or supporting infrastructure. More generally, reproducibility has become a significant concern across computational research, motivating efforts to preserve environments, workflows, and experimental artifacts.

Lean 4 deliberately prioritizes active development over strict backward compatibility. This design choice enables rapid improvement of the language, elaborator, compiler, and ecosystem, and it also introduces challenges for long-term preservation of research artifacts. Experiments may become difficult to reproduce as toolchains evolve, while projects that maintain local modifications or forks can gradually diverge from upstream development. Understanding which factors influenced a historical result can therefore become increasingly difficult over time.

Existing work within the Lean ecosystem has primarily focused on theorem proving, formalization, library development, and language infrastructure. Comparatively less attention has been devoted to ecosystem-level experimentation, governance, and reproducibility infrastructure. In particular, there is limited tooling dedicated to preserving experiment context, enforcing architectural boundaries, and managing ecosystem research independently of modifications to Lean itself.

To address these challenges, we present xLaDe (eXperimental Lean 4 Advanced Development Ecosystem), an ecosystem framework designed to support reproducibility, governance, and structured experimentation around Lean 4. Rather than modifying Lean or maintaining a fork, xLaDe operates

as an orchestration layer surrounding existing Lean projects. The framework records execution metadata, manages experiments through a defined lifecycle, enforces kernel-immutability policies, and provides tooling for validation, verification, and environment diagnostics.

The primary contributions of this paper are:

1. The design and implementation of xLaDe, a Lean 4 ecosystem framework focused on reproducibility and structured experimentation.
2. A kernel-immutability model that enforces a strict separation between Lean's trusted computing base and ecosystem-level experimentation.
3. An experiment framework supporting lifecycle management, policy enforcement, metadata collection, and reproducible execution.
4. A repository governance methodology including a machine-checked experiment schema, lifecycle states, and CI-enforced kernel-immutability policy, illustrated through two demonstration cases of external Lean project integration.

The remainder of this paper is organized as follows. Section 2 discusses the design of xLaDe. Section 3 presents the security and trust model. Section 4 describes the implementation. Section 5 illustrates the framework's operation. Section 6 discusses limitations and future directions. Section 7 concludes the paper.

2 xLaDe Design

xLaDe is designed as an ecosystem layer surrounding Lean 4 rather than a modification of Lean itself. The framework emphasizes kernel immutability, reproducibility, and reversible experimentation while preserving clear boundaries between Lean, ecosystem tooling, and external projects. This section describes the design goals, architecture, and policy mechanisms that guide the development of xLaDe.

2.1 Design Goals

xLaDe was created to investigate whether ecosystem-level challenges surrounding Lean 4 can be addressed without modifying Lean itself. The framework focuses on two specific problems: long-term reproducibility of Lean projects and kernel drift in Lean-based ecosystems. As Lean evolves, proofs and experiments that compile under one toolchain version may fail under another. At the same time, projects that directly modify Lean or maintain long-lived forks can gradually diverge from upstream, making mainte-

nance and diagnosis increasingly difficult. xLaDe approaches these problems through ecosystem tooling rather than language modification.

The framework is guided by three foundational design principles. First, the Lean kernel is treated as immutable infrastructure. Experiments, policies, and tooling operate strictly above Lean's trusted computing base, ensuring that all experimental effects are attributable to ecosystem-level decisions rather than language modifications. Second, experiments are treated as first-class research artifacts. Each experiment must define a research question, enforcement mechanism, scope, reversibility plan, and exit criteria. Third, reversibility is considered a fundamental requirement. Experiments must be removable without affecting unrelated repository state, allowing ideas to be evaluated and abandoned safely when necessary.

Rather than pursuing backward compatibility, xLaDe prioritizes reproducibility. The objective is not to guarantee that old code will continue working indefinitely, but to preserve enough information to reconstruct the environment in which an experiment originally executed. This distinction allows Lean to evolve while preserving the ability to revisit historical research artifacts.

2.2 System Architecture

xLaDe is implemented as an ecosystem layer surrounding Lean 4. The framework does not replace Lean, modify its semantics, or provide an alternative theorem prover. Instead, it acts as an orchestration layer that coordinates experiments, policies, tooling, metrics collection, and reproducibility infrastructure while interacting with Lean exclusively through standard interfaces.

Architecturally, the repository is divided into stable and experimental components. The stable component consists primarily of the `lean-core/` submodule, which contains Lean itself and serves as an immutable reference baseline. Continuous integration workflows enforce this boundary and reject modifications automatically. Experimental components include the CLI implementation, policies, experiments, scripts, metrics, documentation, and operational modes. These components are intentionally designed to evolve over time as new research questions emerge.

The primary user-facing component is the `xlade` command-line interface. The CLI orchestrates experiment execution, mode management, environment diagnostics, policy enforcement, and metrics collection. A dedicated Lean integration layer acts as the sole interface between the Python tooling and the Lean toolchain, ensuring that all interactions with `lake` and `lean` occur through a controlled boundary. Experiments execute through this or-

chestration layer and record structured metadata for later inspection and analysis.

This architecture allows ecosystem experimentation to occur without introducing modifications to Lean itself. As a result, experimental outcomes remain attributable to workflow, tooling, and governance decisions rather than changes in proof semantics or compiler behaviour.

2.3 Policies and Reproducibility

Policies in xLaDe are explicit, documented constraints that operate at the repository and orchestration level. They are designed to protect architectural boundaries, support reproducible experimentation, and provide enforceable governance mechanisms without modifying Lean semantics. Policies may be enforced through scripts, continuous integration workflows, or lightweight Lean-based validation mechanisms. All policies are required to be transparent, reviewable, and reversible.

Several policies originate as ecosystem experiments. New ideas are introduced experimentally, evaluated through observation and metrics, and may later be promoted into reusable tooling or governance mechanisms. This lifecycle prevents premature standardization while preserving historical context for future researchers. Experiments progress through four lifecycle states: draft, active, abandoned, and promoted. The lifecycle model encourages disciplined experimentation while ensuring that unsuccessful ideas remain documented rather than disappearing silently.

Reproducibility is treated as a core architectural objective. Every experiment records metadata describing its execution environment, including the active mode, Lean toolchain version, and execution timestamp. Experiments declare their required toolchains explicitly, and build environments are reconstructed using pinned dependencies and versioned configuration files. This approach allows xLaDe to preserve sufficient context for future re-execution even as Lean continues to evolve.

By combining policy enforcement, experiment lifecycle management, and structured metadata collection, xLaDe attempts to create an ecosystem in which research artifacts remain traceable, reviewable, and reproducible over long periods of time.

3 Security

Security in xLaDe is based on transparency, verifiability, and minimal trust assumptions. Rather than assuming that hosting platforms, mirrors, or

distribution channels are inherently trustworthy, the framework explicitly documents trust boundaries and provides mechanisms for independent verification. The security model is designed to support the long-term reproducibility and integrity goals of the project rather than to function as a general-purpose security framework.

3.1 Threat Model

The primary assets protected by xLaDe are Lean kernel integrity, semantic stability, reproducibility, distribution integrity, and documented trust boundaries. The framework assumes that experimental results should be attributable to ecosystem-level decisions rather than modifications to Lean's trusted computing base.

The threat model considers several classes of attacks and failures. These include accidental modification of the Lean kernel, repository tampering, mirror compromise, DNS hijacking, TLS certificate authority compromise, workflow manipulation, and misleading documentation. Particular emphasis is placed on preventing silent semantic divergence, where ecosystem tooling could alter Lean behaviour without users being aware of the change.

Several threats are explicitly outside the scope of the project. These include vulnerabilities in Lean itself, compromised operating systems, malicious contributors with legitimate repository access, supply-chain attacks affecting external dependencies, and denial-of-service attacks. The project deliberately documents these limitations rather than claiming protection against them.

3.2 Decentralization

xLaDe adopts a decentralized distribution strategy intended to reduce dependence on any single platform. The project maintains a primary repository together with multiple mirrors hosted on GitLab, Codeberg, Bitbucket, Gitea, and SourceForge. Mirrors improve availability, reduce single-platform dependency, and provide alternative access paths when individual services become unavailable. Mirrors are not treated as authoritative sources and users are encouraged to verify repository consistency across sources.

In addition to conventional hosting platforms, xLaDe maintains an official Tor onion service that functions as a first-class project website rather than a fallback access mechanism. Because onion addresses are derived from cryptographic keys, the address itself serves as a form of identity and reduces reliance on DNS infrastructure and certificate authorities. This design pro-

vides resistance against DNS hijacking, TLS compromise, and certain forms of platform-level censorship.

3.3 Availability

Availability is achieved through redundancy rather than trust in any individual provider. Project source code, releases, documentation, and verification materials are distributed across multiple repositories, mirrors, and web interfaces. The project additionally supports torrent-based distribution to reduce dependence on centralized infrastructure. The objective is to ensure that access to research artifacts remains possible even if individual services become unavailable or restricted.

The onion service also contributes to availability by providing a DNS-independent access path. This approach is particularly relevant for users operating in environments where access to conventional hosting platforms may be restricted or monitored.

3.4 Verification

Verification is a central component of the xLaDe security model. The project publishes SHA256 checksums for release archives together with cryptographic identity material including PGP and SSH public keys. Cryptographic keys used for authentication and signing are separated by purpose and are dedicated exclusively to the xLaDe project. This separation reduces cross-project trust dependencies and limits the impact of key compromise. Verification information is distributed through multiple official sources to reduce reliance on a single platform.

Within the framework itself, the most important verification boundary is the protection of the Lean kernel. The `lean-core/` submodule is treated as immutable infrastructure and may not be modified by experiments, contributors, or operational modes. Automated scripts and continuous integration checks detect modifications and reject changes that violate this policy. By enforcing a strict separation between Lean's trusted core and ecosystem-level tooling, xLaDe attempts to ensure that experimental results remain attributable to ecosystem design choices rather than changes in proof semantics.

4 Implementation

xLaDe is actively developed and maintained, with one release per month. As a result, the implementation described in this paper may lag behind the latest version of the project. In this section, we describe the implementation of xLaDe as of v1.7.1, our custom release based on v1.7.0. Since v1.7.1 contains only publication-related refinements and no user-visible CLI changes, screenshots shown in this paper display the version identifier of v1.7.0.

4.1 CLI Tool

xLaDe is implemented as a Python-based command-line application and distributed as a standard Python package. The project requires Python 3.14 or later and exposes a single executable command, `xlade`, through the Python packaging system. The CLI serves as the primary user interface and provides access to experiment management, policy enforcement, environment validation, and reproducibility tooling.

The command-line interface follows a modular subcommand architecture. A central dispatcher parses user input and routes execution to dedicated command modules such as `run`, `doctor`, `mode`, `status`, `metrics`, `check`, and `validate`. This separation allows individual commands to evolve independently while maintaining a consistent user experience.

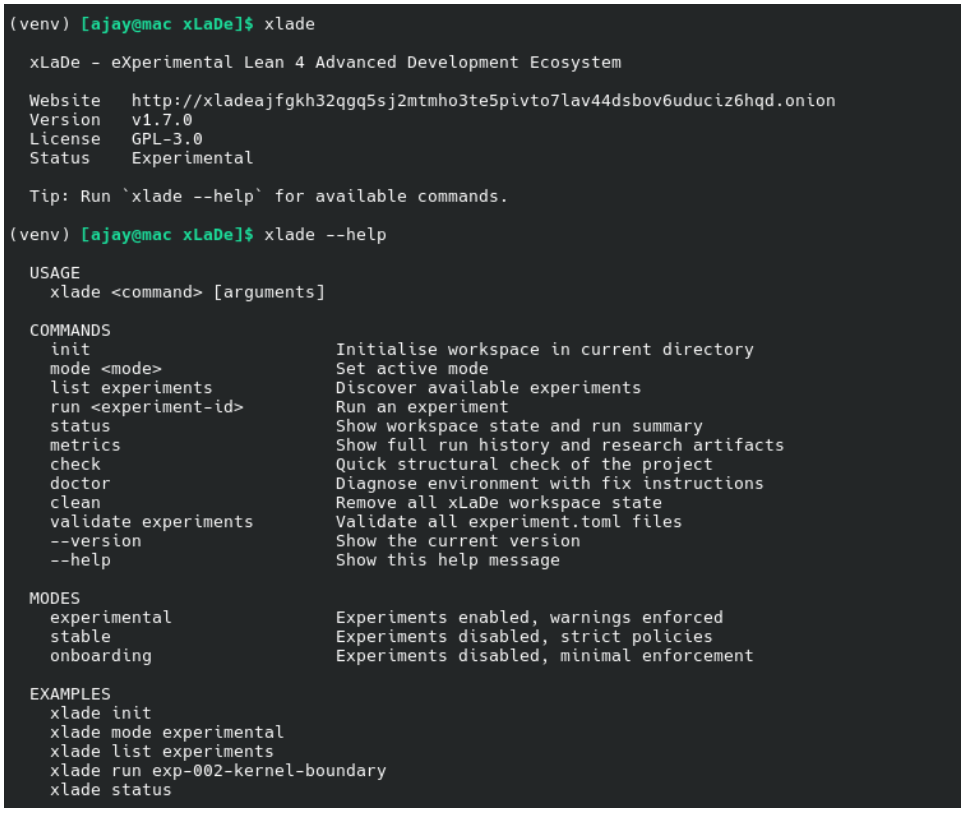


Figure 1: xLaDe CLI

The `run` command is responsible for experiment execution. Before an experiment is launched, xLaDe verifies workspace state, validates experiment metadata, checks mode restrictions, and records execution context including timestamp, toolchain version, and active mode. Results are stored as structured JSON artifacts, creating an audit trail that can be inspected later for reproducibility purposes.

The `doctor` command provides automated environment diagnostics. It generates actionable remediation instructions for missing dependencies, uninitialized submodules, and configuration errors along with reporting failures. This approach reduces onboarding complexity and assists users in constructing valid experimental environments.

xLaDe supports three operational modes: `experimental`, `stable`, and `onboarding`. Modes determine the level of policy enforcement and exper-

iment availability. This mechanism allows the same framework to support research, validation, and educational workflows without requiring modifications to project structure or source code.

The CLI is intentionally designed as a thin orchestration layer. Experiment logic, policy definitions, metadata management, and Lean integration remain separated from the user interface. This separation simplifies maintenance and enables future extension of the framework without changing the external command interface.

4.2 Modes

xLaDe introduces a mode system that allows the ecosystem to operate under different development contexts without modifying Lean itself. A mode represents an orchestration-level configuration that influences experiment activation, policy enforcement, and user-facing behavior while preserving Lean's kernel, compiler, and proof semantics. The design goal is to support diverse workflows ranging from education and onboarding to active research and long-term validation.

The framework currently provides three modes: `onboarding`, `experimental`, and `stable`. Each mode expresses a different intent and determines how ecosystem tooling behaves. Modes are explicitly selected by users and can be changed without rebuilding Lean or modifying project source code.

The `onboarding` mode is designed for new users and educational settings. It minimizes policy enforcement, prioritizes human-readable guidance, and focuses on ease of exploration. This mode is intended to reduce the barrier to entry for first-time Lean users and new contributors.

The `experimental` mode enables active ecosystem experimentation. Selected experiments may be executed, policies may emit warnings, and behavior may evolve as new research ideas are evaluated. This mode is intended for researchers, tooling developers, and contributors investigating new workflows. No backward compatibility guarantees are provided in this mode.

The `stable` mode prioritizes reproducibility and predictability. Ecosystem experiments are disabled, policy enforcement becomes strict, and behavioral changes are minimized. This mode is intended for validation, review, and long-term usage scenarios where stability is preferred over experimentation.

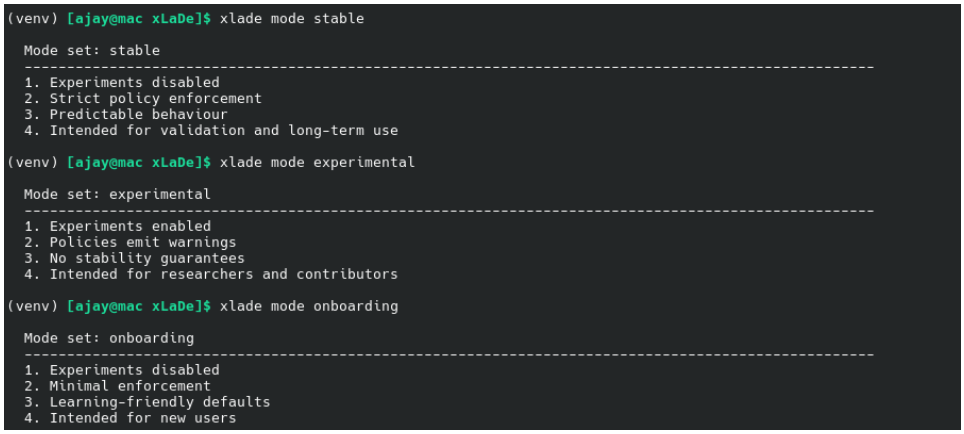


Figure 2: Different modes in xLaDe.

Experiments are activated selectively according to the active mode. By design, experimental functionality is isolated from stable workflows, ensuring that research activities do not affect reproducible or production-oriented environments. This separation allows xLaDe to support experimentation while maintaining clear operational boundaries.

A central design principle of the mode system is that modes influence ecosystem behavior rather than Lean itself. They do not modify the Lean kernel, alter proof semantics, or introduce language-level changes. Instead, they provide an explicit mechanism for controlling experiment activation and policy enforcement while preserving kernel immutability.

4.3 Tooling and Automation

xLaDe employs a collection of automation scripts that reduce manual effort in experiment execution, policy enforcement, and repository maintenance. Rather than relying on informal procedures, common tasks are encoded as executable artifacts that can be repeatedly applied across different environments.

Several scripts are dedicated to policy enforcement. For example, documentation coverage checks verify the presence of required documentation throughout experiments, modes, and policy directories. Similarly, kernel boundary verification ensures that the Lean kernel remains untouched, enforcing xLaDe's design principle that ecosystem experiments must not modify Lean itself.

Automation is also used to execute external projects. Experimental wrappers invoke `lake build` on independent Lean repositories while recording execution outcomes and preserving separation between xLaDe and the wrapped project. This approach allows ecosystem-level experimentation without requiring modifications to the target repository.

In addition to currently deployed tooling, xLaDe includes planned developer assistance utilities. One example is a prototype error-hinting tool intended to augment Lean compiler output with additional diagnostic guidance. Although not yet integrated into the primary workflow, such tools illustrate future directions for improving usability and contributor experience.

The automation layer serves two purposes: reducing operational complexity and increasing reproducibility. By encoding procedures as scripts rather than relying on manual execution, xLaDe aims to make experimental workflows more consistent and repeatable.

4.4 Kernel Protection

A central design requirement of xLaDe is that ecosystem experimentation must not modify Lean's trusted computing base. To enforce this requirement, xLaDe includes a dedicated continuous integration workflow (`kernel-protection.yml`) that verifies the integrity of the `lean-core/` submodule. Any modification to files within this directory causes the workflow to fail, preventing changes from being merged without explicit review.

Unlike documentation-only policies, kernel immutability is therefore enforced automatically rather than relying on contributor compliance. The workflow provides a machine-checkable guarantee that experiments, policies, and ecosystem tooling remain separated from Lean's kernel implementation. This enforcement mechanism operationalizes the kernel-immutability principle described throughout the framework and serves as a repository-level safeguard against accidental semantic drift.

To further validate kernel immutability, xLaDe includes Experiment 2 (EXP-002: Kernel Boundary Violation Detection), which evaluates whether repository-level policies can automatically detect modifications to Lean's trusted computing base. The experiment implements a script-based kernel integrity check that inspects changes to the `lean-core/` directory. If any modification is detected, the check terminates with a failure, preventing the experiment from proceeding and reporting a kernel boundary violation.

The experiment demonstrates that architectural boundaries can be enforced without modifying Lean itself. By operating entirely at the reposi-

tory level and remaining independent of the Lean build configuration, the enforcement mechanism preserves the trusted computing base while providing a machine-verifiable guarantee that kernel modifications are detected automatically. This complements the continuous integration workflow by experimentally validating the kernel-immutability principle within xLaDe's policy framework.

4.5 Tests

xLaDe employs an automated test suite containing 104 tests as of v1.7.1, implemented using the `pytest` framework. The objective of the test suite is to verify the correctness of command-line operations, experiment execution, metadata generation, and Lean integration while remaining independent of user-specific environments.

The test suite covers the primary user-facing commands such as initialization of the workspace, mode selection, experiment discovery, status reporting, metrics collection, project validation, and environment diagnostics. These tests verify both successful execution paths and common failure scenarios such as corrupted state files, missing dependencies, invalid configurations, and uninitialized workspaces.

Core functionality is tested independently from external Lean installations whenever possible. Mocking and fixture-based approaches are used to validate execution behavior, version detection, script invocation, and error handling without requiring a fully configured Lean environment. This improves portability and enables continuous validation across different development systems.

Experiment-specific tests validate the structure and execution requirements of ecosystem experiments. These tests verify experiment metadata, submodule availability, toolchain declarations, execution scripts, and expected repository layouts. Additional tests confirm that experiment execution results are recorded correctly within xLaDe's metrics and reproducibility infrastructure.

The testing strategy emphasizes reproducibility and fault isolation. By validating both normal and exceptional execution paths, xLaDe attempts to detect regressions early and maintain consistency across evolving experimental workflows.

4.6 Experiment Schema Validation

Experiments within xLaDe follow a standardized structure defined by a shared machine-readable schema. Each experiment contains metadata describing its identifier, lifecycle state, execution mechanism, scope, documentation, and supporting artifacts. The schema provides a consistent representation for experiment discovery, execution, and review across the framework.

As of v1.7.1, all five experiments included in the repository conform to the same schema and are validated automatically through repository tooling and continuous integration workflows. This validation process ensures that experiments remain structurally consistent and that required metadata is present before experiments can be considered valid framework artifacts. By enforcing a common representation, xLaDe treats experiments as governed ecosystem objects rather than ad hoc collections of scripts and documentation.

5 Experiments

The experimental framework is a central component of xLaDe. The demonstrations in this section are intended to illustrate the operation of the framework's architecture and policy mechanisms described in Sections 2–4, rather than to provide a comprehensive empirical evaluation of reproducibility guarantees across the Lean ecosystem.

5.1 Experimental Methodology

Experiments in xLaDe are treated as first-class research artifacts. Unlike traditional theorem-proving experiments that focus on proving new mathematical results or extending the Lean language, xLaDe experiments investigate ecosystem-level questions involving tooling, workflows, reproducibility, and project maintenance. Experiments operate strictly outside Lean's trusted computing base and do not modify the Lean kernel, compiler, or proof semantics.

Each experiment is represented by a dedicated directory containing documentation, configuration files, and any supporting artifacts required for execution. Experiments are designed to be isolated, reproducible, documented, and reversible. Every experiment must explicitly define its identifier, lifecycle state, execution mechanism, allowed modes, and scope of influence. This information is recorded in a machine-readable descriptor

that allows experiments to be discovered and executed through the xLaDe command-line interface.

xLaDe adopts a lifecycle model consisting of four states: `draft`, `active`, `abandoned`, and `promoted`. Draft experiments represent early proposals, active experiments are available for evaluation, abandoned experiments are preserved for historical context, and promoted experiments represent ideas that have matured into reusable policies or tooling components. This lifecycle enables experimentation without compromising repository stability.

Experiment execution is controlled by the xLaDe mode system. By default, experiments are enabled only in Experimental Mode and remain disabled in Stable and Onboarding modes. This separation prevents experimental behavior from affecting reproducible or production-oriented workflows while still allowing controlled evaluation of new ideas.

The framework records metadata associated with each experiment execution, including timestamps, execution status, active mode, and Lean toolchain information. These records are stored as reproducibility artifacts that allow experiment outcomes to be inspected and compared across different environments. Failures are treated as research outcomes rather than repository corruption events, allowing experiments to remain informative even when execution is unsuccessful.

The experiments presented in this paper focus on external Lean project reproduction and compatibility analysis. These experiments were selected because they directly evaluate xLaDe's primary objective: preserving and reproducing Lean ecosystem artifacts without modifying Lean itself.

5.2 EXP-004: External Lean Project Reproduction

The objective of EXP-004 was to evaluate whether xLaDe can integrate, execute, and reproduce an external Lean 4 project without modifying either the project source code or the Lean kernel. The experiment investigates the feasibility of treating external Lean repositories as reproducible research artifacts managed through the xLaDe ecosystem. The research question was whether an external Lean 4 repository could be incorporated as a Git submodule, executed through the xLaDe framework, and produce reproducible results while preserving complete separation between the framework and the target project.

The target repository used in this experiment was `exp-004-project-proof-1`[8], a minimal Lean 4 proof repository developed specifically as a controlled test environment. The repository contains a small collection of arithmetic and logical proofs. The project has no external dependencies

beyond the Lean toolchain and does not depend on Mathlib or other third-party libraries. This intentionally minimal design allows the experiment to focus on reproducibility and project integration rather than theorem complexity.

The repository was integrated into xLaDe as a Git submodule located at `experiments/exp-004-project-proof-1/exp-004-project-proof-1`. The experiment was implemented as a script-based policy experiment that executes `lake build` within the external project directory. Compilation success was used as the primary success criterion. A successful execution required all proofs to compile and the build process to terminate with exit code zero. Any non-zero exit status was considered a failure. The experiment was enabled exclusively in Experimental Mode and executed using the toolchain specified by the framework. The execution in terminal is shown below:



Figure 3: Successful execution of EXP-004 using the xLaDe framework.

The results suggest that xLaDe can wrap and execute external Lean projects while preserving project isolation and kernel immutability. No modifications were required to the external repository, and no changes were made to Lean itself. Although the experiment used a deliberately small project, it is consistent with the core architectural assumption underlying xLaDe: external Lean repositories can be treated as reproducible ecosystem artifacts and executed through a common orchestration framework without requiring invasive integration mechanisms.

5.3 EXP-005: Lean4 Courses Compatibility Study

The objective of EXP-005 was to evaluate whether xLaDe could reproduce and execute a Lean 4 repository that was developed independently of the

framework. Unlike previous experiments, the target project was neither authored by the xLaDe team nor designed for integration with the framework. The experiment therefore served as a stronger evaluation of ecosystem-level reproducibility by testing a repository with no prior social, organizational, or technical relationship to xLaDe.

The selected repository was `lean4-courses`, an educational Lean 4 course created by Danny Willems[9]. The repository contains a collection of 32 modules covering introductory and intermediate Lean concepts. The project has no dependencies beyond the Lean standard library and does not require Mathlib or other third-party packages. This characteristic made it suitable for evaluation in a constrained environment where large dependency downloads were impractical.

The repository was integrated into xLaDe as a Git submodule and wrapped as an experimental artifact. The experiment was configured as a script-based policy experiment whose execution mechanism consisted of invoking `lake build` within the external repository. Successful compilation of all modules was used as the primary success criterion.

```
(venv) [ajay@mac xLaDe]$ xlade run exp-005-lean4-courses

  Running experiment:  exp-005-lean4-courses
  Mode:                experimental
  Toolchain:           leanprover/lean4:v4.30.0
  Timestamp:           2026-07-14 10:17:37
  ------------------------------------------------------------------------------------------------------------
  xLaDe EXP-005: Lean4 Courses
  ------------------------------------------------------------------------------------------------------------
  [info]   Project: experiments/exp-005-lean4-courses/lean4-courses
  [info]   Running: lake build
  ------------------------------------------------------------------------------------------------------------
  info: downloading https://releases.lean-lang.org/lean4/v4.30.0/lean-4.30.0-linux.tar.zst
  info: installing /home/ajay/.elan/toolchains/leanprover--lean4---v4.30.0
  i [2/4] Replayed Solutions
  info: 0000-startup/Solutions.lean:17:0: rfl : double 5 = double 5
  info: 0000-startup/Solutions.lean:29:0: true
  info: 0000-startup/Solutions.lean:30:0: false
  info: 0000-startup/Solutions.lean:42:0: 1024
  info: 0000-startup/Solutions.lean:48:0: "olleh"
  info: 0000-startup/Solutions.lean:57:0: 55
  i [3/4] Replayed Examples
  info: 0000-startup/Examples.lean:11:0: 42 : Nat
  info: 0000-startup/Examples.lean:12:0: "hello" : String
  info: 0000-startup/Examples.lean:13:0: Bool.true : Bool
  info: 0000-startup/Examples.lean:15:0: 5
  info: 0000-startup/Examples.lean:16:0: "hello world"
  info: 0000-startup/Examples.lean:19:0: "Hello, Lean 4!"
  info: 0000-startup/Examples.lean:24:0: 10
  info: 0000-startup/Examples.lean:57:0: def Course0000Examples.double : Nat → Nat :=
  fun n => 2 * n
  info: 0000-startup/Examples.lean:63:0: 'Course0000Examples.two_eq' does not depend on any axioms
  info: 0000-startup/Examples.lean:76:0: 3628800
  Build completed successfully (4 jobs).
  ------------------------------------------------------------------------------------------------------------
  [pass]   lake build succeeded. 32 modules compiled.
  ------------------------------------------------------------------------------------------------------------
  Status: success
```

Figure 4: Successful execution of EXP-005 using the xLaDe framework.

The results illustrate the intended operation of xLaDe: ecosystem-level

reproducibility without modification of Lean's trusted core. The framework neither altered the Lean kernel nor injected framework-specific code into the target repository. The repository owner was not involved in the experiment, and the repository itself remained independent of xLaDe. Despite this separation, the project was successfully reproduced and executed through the framework.

6 Discussion

xLaDe is best understood as an architecture and governance contribution rather than a benchmark or measurement study. Its claims rest on the design of enforceable policies and a reproducible experiment methodology, with EXP-004 and EXP-005 serving as illustrative cases rather than comprehensive empirical validation. The experiments presented in this paper provide preliminary evidence that ecosystem-level research around Lean 4 can be conducted without modifying Lean itself. EXP-004 demonstrated that xLaDe can integrate and execute an external Lean repository while preserving complete separation between the framework and the target project. EXP-005 extended this result to a third-party repository that had no prior relationship with xLaDe, demonstrating that external projects can be reproduced and evaluated through a common orchestration framework. Together, these experiments support the central hypothesis underlying xLaDe: meaningful ecosystem experimentation can occur above the Lean kernel rather than inside it.

The experiments reinforce the distinction made throughout this paper between backward compatibility and reproducibility. xLaDe does not attempt to guarantee that old code will continue working indefinitely. Instead, it seeks to preserve sufficient information to reconstruct the environment in which the code originally executed.

The current evaluation includes only a small number of experiments and external repositories. Both evaluated projects were relatively lightweight and did not depend on large external ecosystems such as Mathlib. Consequently, the results should not be interpreted as evidence that all Lean repositories can be reproduced without additional challenges. Furthermore, although xLaDe records metadata required for reproducibility, automatic environment reconstruction remains an active area of development. The framework is also relatively young, making it impossible to evaluate long-term reproducibility claims over many years of Lean evolution.

Future work will focus on expanding the experimental corpus and im-

proving reproducibility infrastructure. Planned directions include evaluation of larger dependency-heavy repositories, compatibility tracking across Lean versions, automated environment reconstruction through `elan`, and support for additional proof assistants beyond Lean 4. Other research directions include experiment analysis, human-readable proof workflows, and broader tooling for ecosystem governance and evaluation. These efforts aim to determine whether the principles explored by xLaDe can generalize beyond Lean and contribute to reproducible research practices across formal verification ecosystems more broadly.

xLaDe should be viewed as an exploratory research platform rather than a finished solution. Its purpose is not to replace Lean, compete with existing tooling, or impose a particular workflow. Instead, it provides a structured environment in which ecosystem ideas can be tested, evaluated, documented, and, when appropriate, proposed upstream. In this sense, the framework is as much a methodology for conducting ecosystem research as it is a software artifact.

7 Conclusion

Lean 4 evolves rapidly, enabling continuous improvements to the language and ecosystem but creating challenges for long-term reproducibility. Proofs, experiments, and research artifacts are often tied to specific toolchain versions, while projects that modify or fork Lean risk diverging from upstream over time. These challenges motivate the need for ecosystem-level approaches that preserve reproducibility without constraining the evolution of the language itself.

In this paper, we presented xLaDe[10], an experimental Lean 4 ecosystem framework focused on reproducibility, governance, and structured experimentation. The framework treats the Lean kernel as immutable infrastructure, records execution metadata for research artifacts, and provides mechanisms for experiment management, policy enforcement, verification, and decentralized distribution. Rather than modifying Lean, xLaDe operates as an orchestration layer surrounding it.

xLaDe's architecture and governance methodology were illustrated through the integration, execution, and reproduction of two external Lean projects, supporting the framework's central design principles. The results suggest that meaningful ecosystem research can be conducted above the Lean kernel while preserving clear architectural boundaries and maintaining reproducibility-oriented workflows.

Although xLaDe remains an experimental project, it provides an initial exploration of how proof assistant ecosystems can be studied through structured, reproducible, and reversible experimentation. We hope that the ideas explored in xLaDe contribute to future discussions on reproducibility, governance, and tooling within Lean and the broader formal verification community.

Acknowledgements and Disclosure

OpenAI Codex was used during the development of the xLaDe project for programming assistance, code suggestions, debugging, implementation support, and post-development code review, including architectural and security review. ChatGPT was used to improve the clarity, grammar, and readability of portions of this manuscript. The research ideas, project design, implementation decisions, experimental methodology, analysis, and the primary content of this manuscript were conceived and written by the authors.

Project Independence

xLaDe is an independent open-source project developed by the authors and is not affiliated with, endorsed by, sponsored by, or officially associated with the Lean 4 project, the Lean FRO, or the Mathlib community. References to Lean 4 throughout this paper describe interoperability with the existing Lean ecosystem and should not be interpreted as implying any official collaboration or endorsement. All views, design decisions, and implementation details presented in this work are solely those of the authors.

References

[1] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn and Jakob von Raumer. *The Lean Theorem Prover*. International Conference on Automated Deduction, 2015.

[2] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, et al. *The Coq Proof Assistant Reference Manual : Version 6.1*. INRIA, 1997.

[3] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer Science & Business Media, 2002.

[4] John Harrison. *HOL Light: An Overview*. Theorem Proving in Higher Order Logics, 2009.

[5] Ulf Norell. *Dependently Typed Programming in Agda*. Advanced Functional Programming, 2009.

[6] Leonardo de Moura and Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*. 28th International Conference on Automated Deduction, 2021.

[7] The mathlib Community. *The Lean Mathematical Library*. Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, 2020.

[8] Lakshit Singh Bisht. *exp-004-project-proof-1*. Accessed: June, 2026. Available at: `https://github.com/LakshitSinghBishtTM/exp-004-project-proof-1`. DOI: 10.5281/zenodo.20746263.

[9] Danny Willems. *Lean4-courses*. Accessed: June, 2026. Available at: `https://github.com/BaDaaS/lean4-courses`.

[10] Lakshit Singh Bisht. *xLaDe: eXperimental Lean 4 Advanced Development Ecosystem*. Accessed: June, 2026. Available at: `https://github.com/LakshitSinghBishtTM/xLaDe`. DOI: 10.5281/zenodo.21311461

Appendices

A xLaDe Repository

High-level organization of the xLaDe repository. Some files and modules are omitted for clarity.

```
xLaDe/
|-- assets/
|   |-- keys/
|   |   |-- pgp.asc
|   |   |-- pgp.txt
|   |   |-- ssh.pub
|   |   |-- ssh.txt
|   |-- torrent/
|-- bin/
|   |-- xlade
|-- docs/
|-- experiments/
```

```
|   |-- exp-001-proof-review/
|   |-- exp-002-kernel-boundary/
|   |-- exp-003-doc-coverage/
|   |-- exp-004-project-proof-1/
|   |-- exp-005-lean4-courses/
|-- lake-manifest.json
|-- lakefile.lean
|-- lean-core/
|-- lean-toolchain
|-- metrics/
|-- modes/
|-- platforms/
|   |-- android/
|   |-- bsd/
|   |-- linux/
|-- policies/
|-- projects/
|-- pyproject.toml
|-- scripts/
|-- security/
|-- tests/
|-- tools/
|-- xLaDe.lean
|-- xlade/
    |-- cli/
    |-- core/
```

B Project Availability

The xLaDe project is publicly available through multiple distribution channels:

- Official website: http://xladeajfgkh32qgq5sj2mtmho3te5pivto7lav44dsbov6uduciz6hqd.onion
- Installation: `pip install xlade`
- GitHub: https://github.com/LakshitSinghBishtTM/xLaDe
- GitLab: https://gitlab.com/LakshitSinghBishtTM/xLaDe
- Codeberg: https://codeberg.org/lakshitsinghbishttm/xLaDe
- Bitbucket: https://bitbucket.org/lakshitsinghbishttm/xlade
- Gitea: https://gitea.com/LakshitSinghBishtTM/xLaDe
- SourceForge: https://sourceforge.net/projects/xlade
- PyPI: https://pypi.org/project/xlade
- Concept DOI: https://doi.org/10.5281/zenodo.18772566

Latest version is also available via torrent, advertised through the official website and repositories. In addition to online distribution, physical releases

on optical media (CD/DVD) and USB flash drives are available upon individual request. Online access to the project is provided free of charge. For physical media, only the cost of the storage medium and shipping is charged.

C Environment

The development, testing and experimentation of xLaDe was done with the following environment:

- CPU: Intel i5-13450HX
- RAM: 24GB RAM DDR5
- GPU: Nvidia RTX 4050 6GB
- OS: Endeavour OS (Arch Linux)
- Kernel: Linux 7.0.x and Linux 7.1.x
- Elan: v4.2.x
- Lean4: Stable

D Example Experiment Descriptor

```
id = "EXP-005"
name = "Lean4 Courses"
type = "script-policy"
status = "active"
allowed_modes = ["experimental"]
lean_toolchain = "leanprover/lean4:v4.30.0"
entry = "scripts/experiments/run-exp-005.sh"
description = "Integration of an external Lean 4 repository
through the xLaDe experiment framework."
success_criteria = "compilation"
project_repo = "https://github.com/BaDaaS/lean4-courses"
project_submodule = "experiments/exp-005-lean4-courses/lean4-courses"
```

E Onion Service

The xLaDe onion service serves as a continuously maintained project website and distribution channel. By using an onion service, xLaDe provides an alternative access path that is independent of conventional DNS infrastructure and certificate authorities.

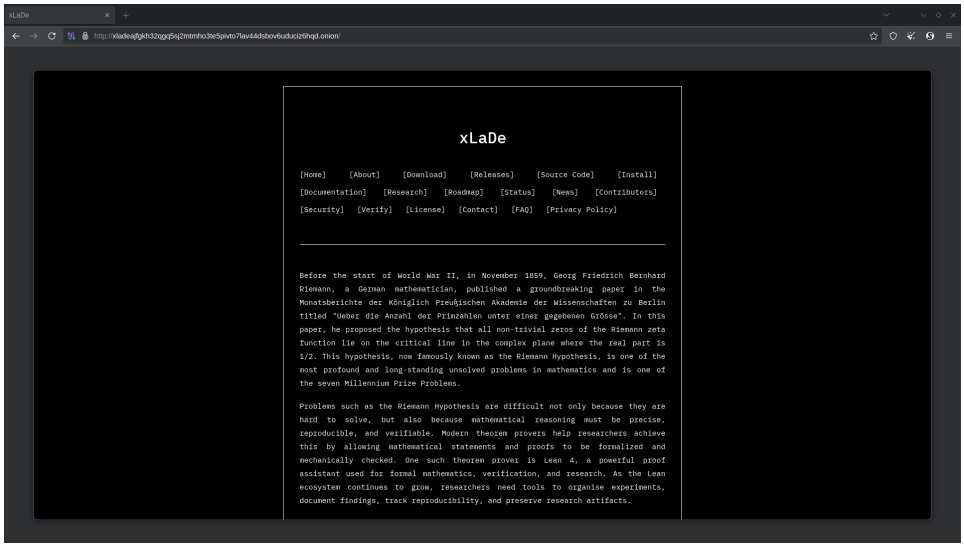


Figure 5: Homepage of the xLaDe onion service.

F Project Contributors

The development of xLaDe benefited from contributions provided by members of the community throughout the project's evolution. The core team gratefully acknowledges the external contributors to the project for their valuable contributions.

- Rishon Pravin
- Aqil Aziz
- Youssef Elganini

G Automated Test Suite

Figure 6 shows the successful execution of the xLaDe automated test suite as of v1.7.1. The framework executed 104 automated tests using the `pytest` framework, with all tests passing successfully and no failures, errors, or warnings reported.

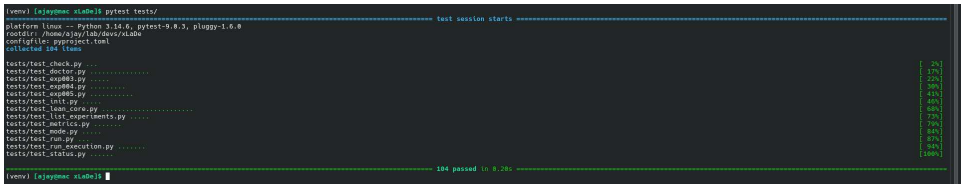


Figure 6: Successful execution of the xLaDe automated test suite.