LEANTRAIL: A VISUAL INTERFACE FOR EXPLORING AND AUDITING MATHEMATICAL FORMALIZATION PROJECTS IN LEAN

A PREPRINT

June 16, 2026

ABSTRACT

Mathematical formalization projects in Lean bring together informal exposition, formalization plans, formal code, and dependencies between declarations. Although development environments support writing and checking code, they provide limited visibility into how the results of a paper are distributed across a repository, and which relationships remain partial, uncertain, or incomplete. We present *LeanTrail*, a visual interface for exploring and auditing the state of Lean formalization projects. The tool constructs an intermediate representation from the paper, the repository, and, when available, the blueprint, connecting mathematical items, plan nodes, Lean declarations, dependencies, and incompleteness indicators. This structure supports coordinated views for navigating the project, tracking results from the paper, comparing planned statements and code, inspecting local graphs, and keeping the analysis grounded in the source code. The central contribution is to make formalization coverage inspectable item by item, distinguishing *Formalized*, *Statement only*, *Partial*, *Blueprint only*, and *Missing* cases, each accompanied by candidate matches and diagnostics. We demonstrate the tool on real formalizations and discuss how this visual traceability supports onboarding, progress tracking, and gap review in Lean projects.

Keywords Lean · Mathematical formalization · Interactive theorem proving · Visual analytics · Formalization coverage · Traceability · Proof engineering

1 Introduction

Mathematical formalization projects in proof assistants such as Lean de Moura and Ullrich [2021] and community libraries such as *mathlib* The mathlib Community [2020] have become increasingly important for verifying results, building formal libraries, and bringing mathematical research and rigorous computational methods closer together. However, following these projects remains difficult, especially for users who were not involved in their construction from the beginning. A formalization is rarely a single artifact: it distributes mathematical reasoning across informal exposition, planning, formal implementation, and intermediate development decisions.

Traditional development environments provide useful mechanisms for reading and editing code, inspecting errors, and navigating files. Even so, a repository-centered view alone is limited for answering conceptual questions about the formalization. Which part of the paper has already been formalized? Does a given theorem appear in the Lean code or only in the blueprint? Does the implementation correspond to the original mathematical statement, or only to a related formulation? Which declarations are still incomplete? These questions cross the boundaries between mathematical text, formalization plan, and formal code, requiring a cross-artifact reading of the project.

This paper presents *LeanTrail*, a visual interface for exploring, inspecting, and monitoring mathematical formalization projects in Lean. The tool organizes the project as an integrated navigation space, connecting items extracted from the paper to blueprint elements, Lean declarations, dependencies, and open points of work. Its central contribution is to make formalization coverage inspectable at the item level. Instead of summarizing progress with a single measure, the interface exposes candidate correspondences and distinguishes five coverage states: *Formalized*, *Statement only*, *Partial*, *Blueprint only*, and *Missing*.

To support this inspection, *LeanTrail* combines multiple coordinated views Roberts [2007]. A structural navigator allows users to switch between files and paper items. A project map shows dependency relations and helps locate central or incomplete regions of the repository. The paper and blueprint views allow mathematical results to be followed through the formalization plan and the Lean code. A focus panel gathers information about the selected element, while the code panel keeps visual exploration grounded in the original formal artifact.

This approach shifts navigation in Lean projects from a purely textual reading toward a visual and relational reading. The goal is not to replace the development environment or to fully automate mathematical judgment about a formalization. Rather, *LeanTrail* organizes evidence, relations, gaps, and uncertainty to support that judgment. By making connections between paper, plan, and code visible, the tool provides a layer of auditability over the formalization process.

This paper makes three contributions. First, we propose a model of a formalization project as a connected collection of heterogeneous artifacts. Second, we present *LeanTrail*, a coordinated-view interface that makes coverage inspectable at the item level by exposing candidate correspondences and classifying them by coverage status and diagnostic relation. Third, we demonstrate the tool on real formalizations, showing how users can follow a result from the paper to the corresponding Lean code and identify gaps, partial correspondences, and pending points in the project.

2 Background and Motivation

Formalizations in Lean combine mathematical writing, programming, and computer-assisted proof. A result presented compactly in a paper may require auxiliary definitions, intermediate lemmas, representational choices, and dependencies across files in order to be expressed formally. This creates a structural distance between the argumentative order of the text, organized by definitions, propositions, lemmas, and theorems, and the operational organization of the repository, distributed across files, namespaces, imports, and proofs. The order of the code rarely follows the order of the paper, and many elements required for the formalization have no explicit counterpart in the text, functioning instead as technical support for the proof.

Blueprints reduce part of this distance by recording an intermediate plan between informal exposition and formal code Massot [2020], Tao [2023]. They indicate what should be formalized, which dependencies are expected, and what remains open. Even so, the user still has to manually coordinate artifacts that evolve at different rhythms. This coordination is especially costly for onboarding and progress tracking. A new collaborator must decide where to start reading, which files are central, which results already have an implementation, and which still depend on future work. These questions are not answered by file navigation alone or by checking that the project compiles, since code may compile while still containing `sorry` or `admit`.

LeanTrail starts from this motivation. The tool treats formalization projects as connected collections of heterogeneous artifacts and makes visible the relations between paper, blueprint, Lean code, dependencies, and open work, supporting a more visual and inspectable reading of the state of the formalization.

3 Related Work

LeanTrail sits at the intersection of mathematical formalization in Lean, blueprint tools, dependency inspection, and visual interfaces for exploring connected artifacts.

Large-scale formalization and blueprints. Lean 4 [de Moura and Ullrich, 2021] and community libraries such as *mathlib* [The mathlib Community, 2020] have made extensive formalized mathematics projects feasible, where the relation between paper, formalization plan, and code is rarely direct (Section 2). The *leanblueprint* tool [Massot, 2020] generates documents and dependency graphs from annotated LaTeX, and recent formalizations, such as the *Polynomial Freiman–Ruzsa conjecture* [Tao, 2023], show the role of these artifacts in coordinating multi-author projects. More recently, *LeanArchitect* [Zhu et al., 2026] has brought blueprint and code closer together by extracting metadata from Lean declarations and synchronizing formal and informal representations. *LeanTrail* is complementary to this line of work: rather than generating or maintaining the blueprint, it offers a visual layer for inspecting coverage between paper, blueprint, and Lean.

Documentation, dependencies, and proof as a navigable artifact. Documentation and structural analysis tools help users navigate formal libraries, but they tend to take the code as their starting point. In the Lean ecosystem, *doc-gen4* [Lean FRO and contributors, 2026] generates HTML documentation from Lean 4 libraries, while the *Lean Atlas* environment and its *Lean Compass* component [Yanahama and Sannai, 2026] support dependency visualization and large-scale semantic review. Work such as *Alectryon* [Pit-Claudel, 2020] also brings text and proof closer together by enabling documents that combine prose, code, and proof states, while tools such as *KnowTeX* [Uskuplu et al., 2025]

explore dependency graphs directly from LaTeX sources. *LeanTrail* shares this concern with navigation and traceability, but changes the main unit of analysis: the starting point is the mathematical item in the paper. Thus, the interface does not ask only which declarations depend on which others, but which results from the paper are *Formalized*, *Statement only*, *Partial*, *Blueprint only*, or *Missing*.

Exploratory visualization of connected artifacts. The organization of the interface is also related to classical principles of exploratory visualization, such as multiple coordinated views [Roberts, 2007] and the combination of overview, filtering, and details on demand [Shneiderman, 1996]. In *LeanTrail*, these principles are applied to the specific context of mathematical formalizations: the project map, paper view, blueprint view, focus panel, and source code remain connected through coordinated selection. This combination allows users to move between global overview and local inspection without losing the link to the original formal artifact.

4 Design Goals

Based on the challenges described above, we define four design goals to guide the construction of *LeanTrail*. These goals position the tool not merely as a Lean file browser, but as a visual interface for understanding the state, coverage, and structure of a mathematical formalization.

G1 — Connect heterogeneous artifacts. Make explicit, within a common interface, the relations between paper, blueprint, Lean files, declarations, proofs, and incomplete points, allowing the same mathematical item to be followed across these artifacts without manually switching between tools.

G2 — Support global overview and detailed inspection. Combine *overview* and *drill-down*: identify the structure of the repository and its central or incomplete regions, and from there move toward specific files, declarations, or items in a progressive navigation from the project as a whole to the corresponding code.

G3 — Make coverage inspectable. Represent coverage at the item level, distinguishing *Formalized*, *Statement only*, *Partial*, *Blueprint only*, and *Missing* states instead of reducing the state of the project to a single metric. This distinction favors an auditable reading of the relations between paper, blueprint, and Lean.

G4 — Expose gaps, uncertainty, and open work. Show not only complete formalizations, but also incomplete proof statements, partial correspondences, blueprint-only plans, ambiguous candidates, and paper items with no reliable link to the code, supporting progress tracking and future development.

5 System Overview

LeanTrail transforms the heterogeneous artifacts of a formalization project into a common, precomputed representation that feeds the coordinated-view interface. The pipeline starts from three inputs — the mathematical paper, the local Lean repository, and, when available, the blueprint files —, processes each one separately, and computes an alignment layer between paper, plan, and code. Figure 1 summarizes this flow. The interface does not infer the structure of the project directly from the source files during interaction; instead, it consumes a representation whose entities and relations can be traced back to their source artifacts.

5.1 Inputs and Project Data Model

Each input is processed by a specific extractor before any alignment. From the **paper**, the system extracts theorem-like environments, labeled mathematical objects, equations, references, sections, and positions in the text. From the **Lean repository**, it indexes files, modules, *imports*, declarations, signatures, source-code ranges, and proof-completeness metadata; a declaration is marked as incomplete when its range contains `sorry` or `admit`. From the **blueprint**, when present, it extracts nodes, labels, statements, dependency edges, and explicit links to Lean declarations.

References between symbols are normalized into a declaration graph, and *imports* into a dependency graph between files. Over these structures, the system computes derived summaries, such as counts of declarations, incomplete proofs, graph degrees, and possible entry points. These artifacts are exported as JSON structures ready for the interface, allowing all views to share the same database.

5.2 Alignment and Coverage Classification

The alignment layer connects items extracted from the paper to possible counterparts in the blueprint and Lean code. For each matchable item, *LeanTrail* retrieves candidates through two routes: a direct *Paper→Lean* route, based on Lean

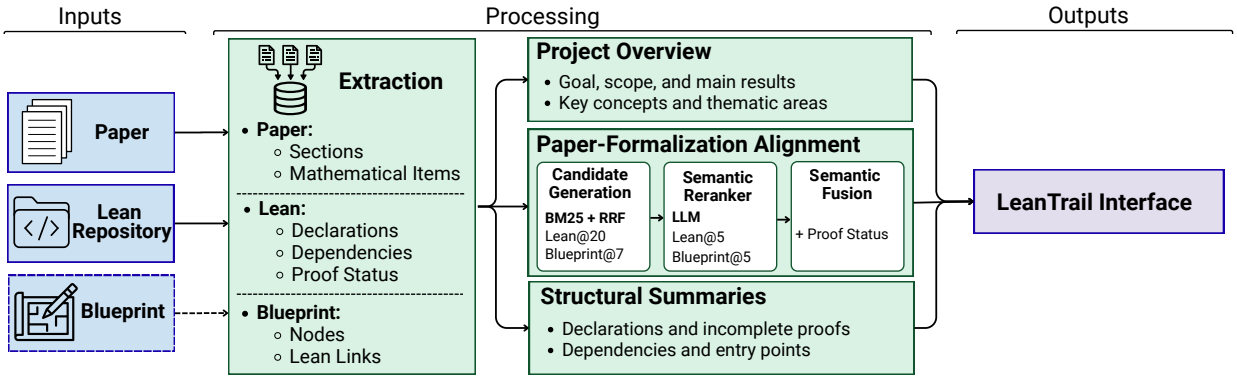

Figure 1: *LeanTrail* pipeline. The system receives a paper, a Lean repository, and, when available, a blueprint; extracts mathematical items, declarations, dependencies, proof status, and blueprint links; generates project summaries and Paper–Lean/Paper–Blueprint alignments; and exports an intermediate representation used by the interface for overview, coverage inspection, and source-level navigation.

declarations, and a *Paper→Blueprint* route, based on blueprint nodes with optional links from those nodes to local Lean declarations. This separation preserves both results planned in the blueprint but not yet implemented and relevant Lean declarations that were not explicitly linked to the plan.

Retrieval is treated as candidate generation, not as a coverage decision. Paper items, blueprint nodes, and Lean declarations are converted into normalized *cards* with fields such as prose, mathematical notation, concepts, identifiers, names, and location. The backend ranks candidates with BM25 [Robertson and Zaragoza, 2009] in each field and combines the rankings using *Reciprocal Rank Fusion* [Cormack et al., 2009]. This stage keeps the top 20 Lean candidates and the top 7 blueprint candidates for each paper item. Over this reduced set, a semantic reranker implemented by a language model classifies the relation to the paper item and keeps the top 5 candidates for each route, distinguishing equivalent, partial, support-only, related-only, or unreliable correspondences. Proof completeness, however, is not judged by the model: it remains a deterministic metadata field extracted from the Lean code.

The final fusion combines the semantic judgment with deterministic metadata to assign one of five coverage states displayed in the interface. *Formalized* means that the paper item has a faithful Lean counterpart whose proof is complete. *Statement only* means that a faithful Lean statement exists, but the corresponding source range contains `sorry` or `admit`. *Partial* means that the best candidate covers only part of the paper item, or corresponds to a weaker or auxiliary formulation. *Blueprint only* means that the item is represented in the blueprint but has no reliable local Lean counterpart. *Missing* means that no reliable Lean or blueprint counterpart was found. Weak evidence does not by itself assign a coverage state, but remains available for inspection. Thus, the language model acts only over previously retrieved candidates, and the final state remains anchored in the original artifacts.

6 Visual Interface

The *LeanTrail* interface is an inspection environment organized into three areas. On the left, the structural navigator (Figure 2A) switches between Lean files and paper items; below it, the focus panel summarizes the selected element and lists links to related artifacts. In the center, three main tabs, *Project map*, *Paper*, and *Blueprint*, provide the tool's analytical views, including a paper-centered workflow that links extracted paper items to Lean code, blueprint nodes, and coverage diagnostics (Figure 3). On the right, the code panel (Figure 2D; Figure 3C) displays the selected Lean file and highlights the relevant declaration. This arrangement makes it possible to explore the project through different paths without losing sight of the source code.

Navigating the Project Structure. The structural navigator (Figure 2A) is the entry point of the interface and operates in two modes. In *Files* mode, users browse the repository's directories and files, which is useful when they already know part of the organization or want to open a specific module. In *Paper* mode, the navigator presents items extracted from the paper, such as definitions, lemmas, propositions, and theorems (Figure 3A), allowing exploration to start from the mathematical text without knowing in advance where each result appears in the code. Search and filters complete the panel: search locates files, declarations, or items by name or text fragment, while filters restrict or highlight the list, for example by marking files that contain `sorry` or `admit`. When an element is selected, the selection propagates to the other regions of the interface, updating the central view, the focus panel, and, when possible, the code panel.

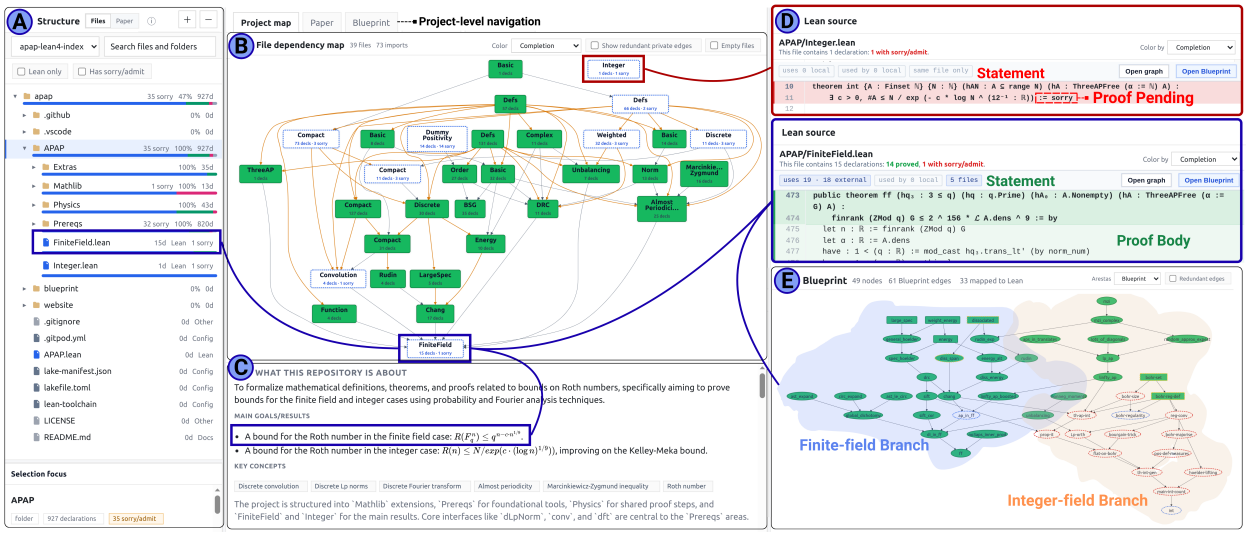

Figure 2: *LeanTrail* interface on the APAP project, focusing on the *Project map* and *Blueprint* tabs. The structural navigator (**A**), project map (**B**), repository summary (**C**), Lean source panel (**D**), and blueprint view (**E**) operate as coordinated views. Together, they connect the repository structure, formal code, and blueprint branches, exposing both complete Lean proofs and pending statements.

Project Map. The *Project map* tab provides an overview of the repository by combining a graph of files and dependencies (Figure 2B) with a project summary panel (Figure 2C). The graph serves as a structural view: users can identify central modules, peripheral regions, and files that concentrate incomplete work before opening individual files. Files with incompleteness indicators can be visually highlighted, making the map a combination of dependency navigation and tracking of the current state of the formalization. Below the graph, the summary panel presents a high-level description of the repository, its main concepts, thematic areas, and possible entry points for reading. This summary complements the visual structure of the graph with an interpretive layer, helping users understand the function of different project regions without manually reconstructing this organization from *imports* and file names. When a node is selected, the interface reveals its direct dependencies and related elements, connecting the global view to the corresponding code.

Paper View. The *Paper* tab presents paper items as interactive elements and is where coverage becomes inspectable. For each item, the interface displays its coverage state together with candidate Lean declarations and blueprint nodes. When there are multiple candidates, users can compare alternatives and inspect the justification for each suggested relation. Figure 3 illustrates this paper-centered workflow: paper items are inspected in context (Figure 3B), linked to Lean source snippets (Figure 3C), explained through the focus panel (Figure 3D), and summarized in a coverage table comparing paper, blueprint, and Lean correspondences (Figure 3E). Thus, exploration starts from the mathematical result in the paper and moves toward its possible formal counterparts. This tab functions both as a coverage view and as an audit tool: partial, uncertain, or unmatched items become starting points for manual review, where users inspect candidates, open the code, and verify whether the relation holds.

Blueprint View. The *Blueprint* tab presents the formalization plan, when available, as a navigable artifact (Figure 2E). Users browse nodes, inspect their statements or descriptions, and check how they relate to paper items and Lean declarations. Unlike the original blueprint, the interface displays the LaTeX statement and Lean code side by side, making it easier to compare the plan with its implementation. This is relevant because the blueprint occupies an intermediate position between paper and code: a paper item with no direct Lean counterpart may exist as a planned node, and a Lean declaration may implement only part of a node. The tab also reveals progress in the plan, allowing users to identify which nodes already have a code correspondence, which remain only planned, and which have uncertain relations. The relation between paper items, blueprint nodes, and Lean declarations is also summarized from the paper side in the coverage table (Figure 3E).

Code Panel. The code panel anchors visual exploration in the Lean source code (Figure 2D; Figure 3C). When a file, declaration, paper item, or blueprint node has a Lean counterpart, the panel opens the corresponding source file and highlights the relevant declaration. It also exposes proof-completeness information, distinguishing complete proofs from declarations whose source range contains `sorry` or `admit`.

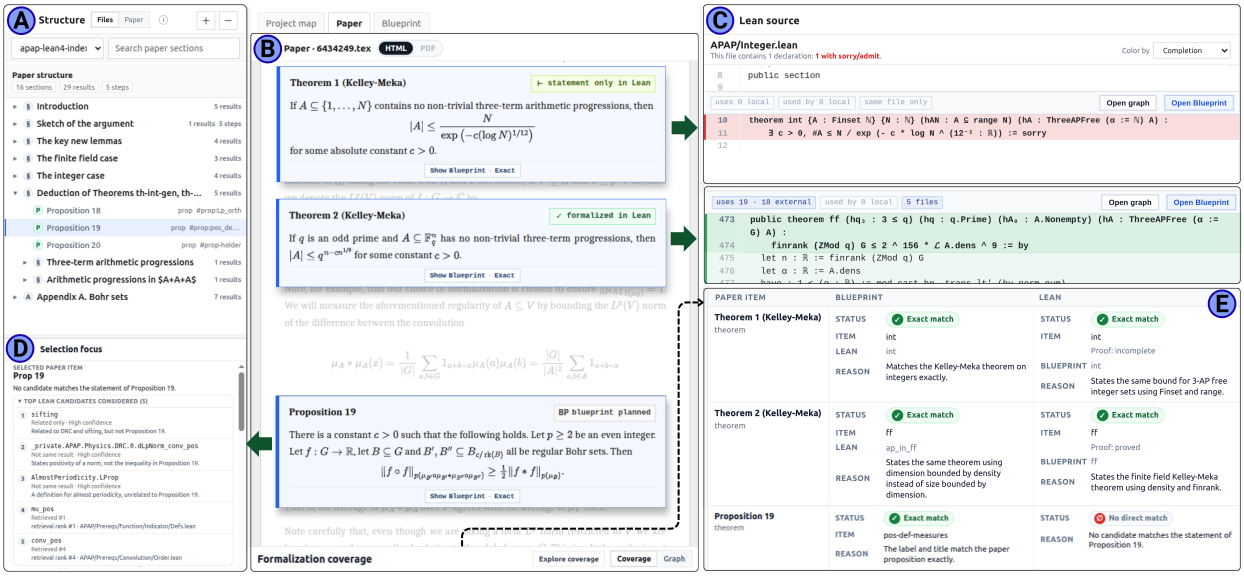

Figure 3: The *Paper* tab in an APAP case study. The structural navigator lists extracted paper items (**A**); selected results are inspected in the paper context (**B**); Lean source snippets show the corresponding declarations and their proof status (**C**); the focus panel exposes retrieved candidates and diagnostic explanations (**D**); and the coverage table summarizes the relation between paper items, blueprint nodes, and Lean declarations (**E**).

Coordinated Selection and Source Grounding. The views are connected by a common selection mechanism. When a file, paper item, blueprint node, or declaration is selected, the interface updates the central view, the focus panel (Figure 3D), and, when possible, the code panel. The focus panel consolidates information about the active element, including type, text, correspondence state, candidates, signature, dependencies, or links to other artifacts, depending on the selected item. Together, the focus panel and the code panel support different trajectories: starting from the project map and moving to the corresponding source file, starting from a paper theorem and moving to its Lean candidates, or starting from a blueprint node to verify whether it already appears in the repository. The paper-centered path is shown in Figure 3: a user can start from a paper item, inspect its candidate Lean and blueprint correspondences, and then verify the highlighted source declaration.

7 Usage Scenario

We consider APAP (Arithmetic Progressions, Almost Periodicity), an ongoing Lean formalization of bounds for Roth numbers in the finite-field and integer cases.[1] The scenario follows a collaborator who is not yet familiar with the repository and uses LeanTrail to understand how the formalization is distributed across the paper, blueprint, and Lean code, which results already have complete Lean proofs, and where there are still concrete points to contribute.

The reading begins with the *Project map* tab (Figure 2). The repository summary (C) describes the project and identifies its central goal: proving the two bounds. In the tree (A), `FiniteField.lean` and `Integer.lean` appear at the root level of the `APAP` folder, alongside the infrastructure subfolders (`Prereqs`, `Physics`), indicating that they correspond to the two goals. The dependency graph (B) refines this reading: `FiniteField.lean` occupies a downstream position — it depends on several local components (almost periodicity, discrete convolution, Fourier, L^p norms) and is not imported by any other file —, which marks it as the endpoint of the finite-field branch. `Integer.lean`, by contrast, appears isolated, with no local imports and a single declaration. The *Blueprint* tab (E) confirms the contrast: the finite-field branch is almost entirely connected to Lean code, with one localized gap, while the integer branch appears mostly as a plan, which is consistent with the isolation of `Integer.lean` in the graph.

This reading is repository-centered. Since APAP formalizes an already existing paper, however, the paper functions as the specification of the project, and measuring progress means checking which of its results already have a counterpart in the plan or in the code — a perspective that artifact-based reading does not offer directly. The collaborator then moves to the *Paper* tab (Figure 3), which starts from the paper and presents each result (A, B) as an item with a coverage state

[1] `https://github.com/YaelDillies/APAP`

The two goals of the project correspond to the two main theorems of the paper, and the tab takes up this contrast at the level of results. *Theorem 2 (Kelley–Meka)*, from the finite-field branch, has an exact correspondence in the blueprint and in Lean (E), with the declaration `ff` shown as a complete Lean proof in the code panel (C): this confirms, from the paper side, the reading of the map. *Theorem 1 (Kelley–Meka)*, from the integer branch, also matches exactly at the statement level, but the declaration `int` has an incomplete proof. This refines the impression left by the map and the blueprint: although the branch appears mostly as a plan, the main result is not missing — the statement has already been translated, and what is missing is the proof —, making concrete a point of contribution previously described only as a "less developed integer branch". Beyond the two main theorems, the interface exposes the remaining results of the paper. *Proposition 19*, for example, illustrates the *Blueprint only* state: it exists as a planned node in the blueprint, but has no direct counterpart in Lean, and the focus panel (D) lists the retrieved candidates and explains why they are related but not equivalent — a gap that file navigation alone would not reveal.

The two readings are complementary. The map gives the global view of where the work is concentrated; the paper, item by item, separates *Formalized* code, *Statement only* results, and *Blueprint only* nodes. For the collaborator, the paper comes to function as an index of progress against the specification: the collaborator starts from a known result, checks whether there is corresponding code, distinguishes a complete proof from an incomplete one, and arrives at concrete points where contribution is possible.

8 Discussion and Future Work

The APAP scenario shows how LeanTrail supports the four design goals: connecting heterogeneous artifacts; switching between global overview and local inspection; making coverage inspectable item by item; and exposing gaps, uncertainty, and open work through the five coverage states. The main implication is to treat the paper as the specification of the project. While repository-centered reading shows *where* the work is concentrated, paper-centered reading shows *what* is still missing with respect to the mathematical results that motivate the formalization. LeanTrail does not automate the mathematical judgment of these correspondences, but organizes candidates, evidence, and uncertainty into a layer of auditability complementary to the development environment, the blueprint, and documentation tools.

Limitations. The evidence presented is still illustrative, based on a usage scenario over an ongoing project, without a user study or comparison with baselines. This limits claims about generalization, readability and scalability in larger projects, and practical impact on onboarding. In addition, coverage depends on candidates retrieved by BM25 with RRF and classified by a language-model reranker, which may yield false negatives, false positives, or relations that are semantically plausible but not faithfully equivalent to the statement in the paper. Extraction also presupposes relatively structured sources, such as theorem-like environments in LaTeX and, when available, a maintained blueprint. Finally, the current representation is precomputed and may grow stale as the repository evolves.

Future Work. We intend to evaluate LeanTrail with formalizers in onboarding, progress-tracking, and gap-review tasks, including comparisons with direct navigation through the repository, documentation, and blueprint. On the technical side, we plan to integrate the representation with a live Lean environment, via LSP and incremental builds, investigate correspondences that take types and notation into account, and make more explicit the distinction between a semantically related candidate and a statement faithful to the paper. We also plan to support interactive corrections, allowing users to confirm, reject, or adjust correspondences and turn those decisions into persistent project metadata. Finally, we intend to study the scalability of the interface on larger projects and subprojects of large-scale libraries such as `mathlib`.

9 Conclusion

We presented LeanTrail, a visual interface for exploring and auditing mathematical formalization projects in Lean, connecting artifacts that are usually inspected separately — paper, blueprint, Lean code, dependencies, and completeness indicators — through coordinated views that support both global project navigation and item-by-item inspection. The APAP scenario illustrates how a paper-centered reading complements repository-centered navigation: while the project map shows *where* the work is concentrated, the paper and blueprint views show, for each result, which of the five coverage states applies. This makes formalization coverage more traceable and auditable while keeping inspection anchored in the source code. LeanTrail does not replace mathematical judgment or the Lean development environment; its contribution is to organize evidence, correspondences, and gaps, making the state of the formalization more legible for onboarding, progress tracking, and review of remaining open points.

References

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. doi:10.1007/978-3-030-79876-5_37.

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381. Association for Computing Machinery, 2020. doi:10.1145/3372885.3373824.

Jonathan C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the 5th International Conference on Coordinated and Multiple Views in Exploratory Visualization*, CMV 2007, pages 61–71. IEEE Computer Society, 2007. doi:10.1109/CMV.2007.20.

Patrick Massot. Lean blueprints. `https://github.com/PatrickMassot/leanblueprint`, 2020. GitHub repository. Accessed: 2026-06-01.

Terence Tao. Formalizing the proof of PFR in Lean4 using Blueprint: A short tour. `https://terrytao.wordpress.com/2023/11/18/formalizing-the-proof-of-pfr-in-lean4-using-blueprint-a-short-tour/`, November 2023. Blog post. Accessed: 2026-06-01.

Thomas Zhu, Pietro Monticone, Jeremy Avigad, and Sean Welleck. LeanArchitect: Automating blueprint generation for humans and AI, 2026.

Lean FRO and contributors. doc-gen4: Document generator for Lean 4. `https://github.com/leanprover/doc-gen4`, 2026. GitHub repository. Accessed: 2026-06-01.

Banri Yanahama and Akiyoshi Sannai. Lean Atlas: An integrated proof environment for scalable human-AI collaborative formalization, 2026. Includes the Lean Compass component.

Clément Pit-Claudel. Untangling mechanized proofs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2020, pages 155–174, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381765. doi:10.1145/3426425.3426940.

Elif Uskuplu, Lawrence S. Moss, and Valeria de Paiva. KnowTeX: Visualizing mathematical dependencies, 2025.

Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pages 336–343. IEEE Computer Society, 1996. doi:10.1109/VL.1996.545307.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi:10.1561/1500000019.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 758–759. ACM, 2009. doi:10.1145/1571941.1572114.