PAPER2LEAN: AGENTIC AUTOFORMALIZATION OF MATHEMATICAL RESEARCH PAPERS AT SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

Formalizing a mathematical research paper requires more than translating isolated theorems: statements and proofs must remain semantically faithful to the source and globally consistent across long-range dependencies. We introduce PAPER2LEAN, an end-to-end agentic framework that converts mathematical research papers into integrated Lean projects. The framework extracts mathematical content into a dependency graph and coordinates specialized agents for statement formalization, semantic checking and repair, proof construction, and project integration. We evaluate PAPER2LEAN on 20 mathematical optimization research papers, using `mathlib` and a shared library of supporting results. All 20 automatically generated projects compile successfully without proof placeholders. Following human review and semantic repair, we introduce OPTPAPER-20, a corpus of 20 corrected paper-level formalizations comprising 494 Lean files and 110,910 lines of Lean code. All released projects pass clean builds, with no unresolved discrepancies identified in the audit. These results demonstrate the feasibility of agentic paper-level formalization while highlighting semantic fidelity as a central challenge beyond proof verification. The source code and dataset are publicly available at `https://anonymous.4open.science/r/Paper2Lean-5A6D/`.

1 INTRODUCTION

Interactive theorem provers such as Rocq (Barras et al., 1997), Isabelle (Nipkow et al., 2002), and Lean (De Moura et al., 2015) have become essential infrastructure for AI for Mathematics (AI4Math), providing formal languages and trusted kernels for constructing and verifying machine-checkable mathematics. Among them, Lean is based on dependent type theory and supports the representation of mathematical statements and proofs as executable, machine-checkable code. Its mathematical library, `mathlib` (mathlib Community, 2020), offers an extensive and continuously growing collection of reusable definitions, theorems, and proof automation across diverse areas of mathematics. Together, Lean and `mathlib` provide a particularly suitable foundation for research-level formalization: once a mathematical result has been faithfully encoded, its proof can be checked by Lean's small trusted kernel and subsequently reused in larger formal developments.

AI for mathematics has advanced rapidly in recent years. In 2026, AI-assisted systems made particularly notable progress in conjecture resolution and formal proof search, including mathematical results checked by Lean (Ju et al., 2026b; Tsoukalas et al., 2026). Large scale efforts also released complete Lean formalizations of major mathematical results, such as Fermat's Last Theorem (Anthropic, 2026). These developments show that LLM agents can contribute not only to discovering or proving individual results, but also to producing reusable, machine-checked mathematical knowledge. The next challenge is to formalize the research papers that contain such results as coherent Lean projects.

Formalizing an entire research paper is substantially different from translating an isolated theorem. First, the mathematical content must be recovered from a PDF, including notation, cross-references, and the structure of definitions and results. Second, the formal statements must remain aligned with the source despite omitted or implicit assumptions and variations in notation. Third, the resulting proofs and declarations must be integrated into one project whose imports, interfaces, and downstream dependencies remain consistent as the formalization evolves.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address these challenges, we introduce PAPER2LEAN, an end-to-end agentic framework for formalizing research papers as integrated Lean projects. Figure 1 provides an overview. PAPER2LEAN first applies OCR and document extraction to identify the mathematical content of a paper and organize it into a structured dependency graph. It then coordinates specialized agents for statement formalization, semantic checking, library API reuse, proof planning and construction, proof refactoring, and library integration. Compilation gates are applied throughout the pipeline, and verified interfaces are frozen before dependent items are processed. This design allows independent items to be handled concurrently while maintaining consistent definitions, statements, proofs, and dependencies across the complete paper.

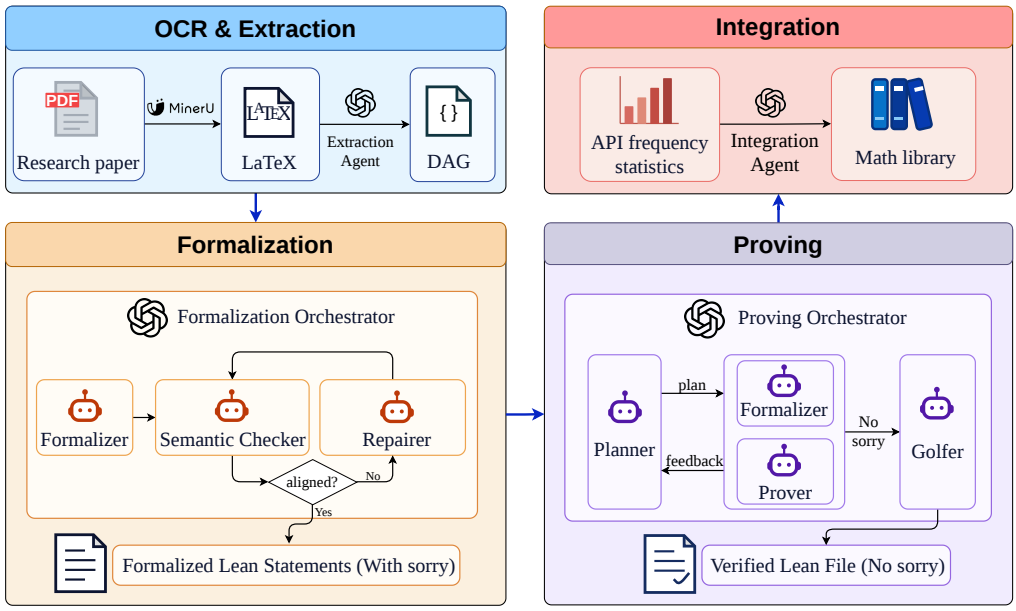


Figure 1: Overview of PAPER2LEAN.

We evaluate PAPER2LEAN on 20 research papers in mathematical optimization. The automatically generated projects all compile successfully and contain no `sorry` placeholders. Experts judged 473 of 494 items to be source-faithful (95.7%). After manual correction, we release the corrected project as OPTPAPER-20, a corpus containing 494 Lean files and 110,910 lines of Lean code. All identified discrepancies were repaired in the released projects.

Our contributions are summarized as follows:

1. We introduce PAPER2LEAN, an end-to-end agentic framework for formalizing mathematical research papers in Lean. Starting from a paper PDF, PAPER2LEAN extracts and organizes mathematical content into a dependency graph, then coordinates specialized agents for statement formalization, semantic checking, proof construction, golfing, and project integration. Compilation-gated execution and frozen interfaces enable parallel processing while preserving semantic fidelity, Lean-checked correctness, and global consistency across the paper.
2. We evaluate PAPER2LEAN on 20 research papers in mathematical optimization. All generated projects compile without proof placeholders. An expert audit finds that 473 of 494 formalized items are source-faithful, while the remaining discrepancies demonstrate that Lean compilation alone does not establish fidelity to the source paper.
3. We release OPTPAPER-20, an expert-refined dataset built from the formalizations of these 20 papers. For each paper, the dataset provides the refined Lean files, corresponding natural language descriptions, and the structured JSON file extracted from the source paper, together with mappings between source paper items and their formal counterparts.

2 Related Work

2.1 Autoformalization and automated theorem proving

LLM-based autoformalization can be broadly divided into statement-level and proof-level formalization. Statement-level methods translate natural language mathematical statements into Lean declarations. They use tools such as `mathlib` retrieval and Lean feedback to improve formalizations and construct high quality datasets for fine tuning, see (Hanyu et al., 2026; Liu et al., 2026c;b). Proof-level methods formalize complete natural language proofs at the project level. ProofFlow (Cabral et al., 2026) organizes dependencies among informal proof steps as a directed acyclic graph (DAG) and formalizes the resulting intermediate lemmas and proofs into a Lean project. Our work instead focuses on formalizing an entire mathematical paper as a single Lean project, guided by the DAG of dependencies among the paper's theorems.

Automated theorem proving assumes a formally specified theorem and aims to construct a machine-checkable proof. Recent neural theorem provers combine LLM generation with premise retrieval, proof search, and Lean feedback, such as LeanDojo and ReProver (Yang et al., 2023). More recent systems, such as DeepSeek-Prover-V2 (Ren et al., 2025) and Kimina-Prover (Wang et al., 2025), improve proof generation through subgoal decomposition, structured reasoning, and reinforcement learning. While these methods achieve strong performance on isolated theorem-proving benchmarks (Zheng et al., 2021; Azerbayev et al., 2023), they generally assume that theorem statements and their surrounding formal contexts have already been provided, rather than constructing a consistent formal development from a mathematical document.

2.2 Agentic formal reasoning

Recent work has explored agentic workflows that coordinate large language models, formal verifiers, retrieval systems, and specialized reasoning modules. Numina-Lean-Agent (Liu et al., 2026a) equips a coding agent with Lean interaction, theorem retrieval, and iterative proof-repair tools, while M2F (Wang et al., 2026) extends agentic formalization to entire documents and projects through dependency-aware statement construction and proof repair. For research-level mathematics, Archon (Ju et al., 2026b) decomposes informal arguments and translates them into multi-file Lean projects using retrieval and verifier-guided refinement. Together, these systems demonstrate the benefits of tool use, task decomposition, and specialized-agent collaboration. Our work focuses on paper-level formalization, where agents must maintain global consistency and resolve long-range dependencies among definitions, theorems, and proofs, rather than solving each theorem in isolation.

3 Paper2Lean

In this section, we describe the main stages of Paper2Lean shown in Figure 1, including OCR and extraction, statement formalization, theorem proving and library integration.

3.1 OCR and extraction

Given a research paper in PDF form, we use MinerU (Wang et al., 2024) to obtain Markdown and LaTeX versions of the document. The extraction agent then reads the entire document and identifies the mathematical units required for formalization, including standing assumptions, definitions, algorithms, lemmas, propositions, theorems, corollaries, and formally relevant remarks. Results cited from external sources are represented explicitly. The agent also introduces definition-gap items when a construction is used by the paper but not explicitly defined, and propagates section-level assumptions to all affected results. The extracted content is stored in a canonical JSON representation. Each item corresponds to one mathematical unit and records its source statement, optional proof text, type, and dependencies. See Appendix A for an example.

Following the approach proposed by Ju et al. (2026a), we represent each paper as a directed acyclic graph (DAG), where each node corresponds to a mathematical unit and directed edges encode dependencies between units. A directed edge from u to v means that the declaration or proof of v is based on u (see Figure 2); consequently, prerequisites appear above the items that depend on them. The upper layers therefore typically contain standing assumptions and basic definitions, while lower

layers contain lemmas and theorems that build on them. The extraction and planning stages validate the references and reject cycles, so every item can be assigned a dependency depth.

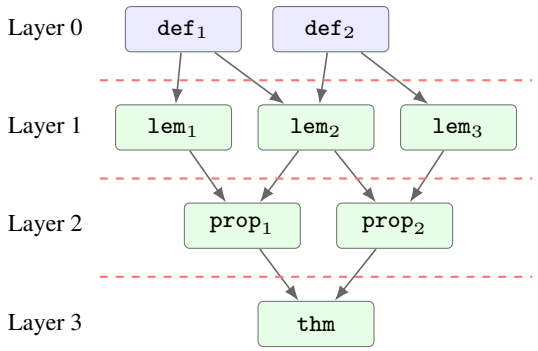


Figure 2: Illustrative DAG for paper-level formalization. The labels indicate item types with `def` for definition, `lem` for lemma, `prop` for proposition, and `thm` for theorem. A lemma may depend on a definition, while a theorem may depend on preceding propositions; dashed lines indicate the layer barriers used by the proving scheduler.

3.2 Statement formalization

The formalization orchestrator coordinates three workers for each item: the **formalizer**, the **semantic checker**, and the **semantic repairer**. For each item, the formalizer first reads the source statement and its dependencies. It then imports the corresponding dependency modules, searches `mathlib` and the shared `Foundation` for relevant APIs, avoiding redundant definitions and improving compatibility with the existing formalization ecosystem. An initial Lean formalization of the original statement is then generated.

Since the generated statement may not fully preserve the intended meaning of its natural language counterpart, it is subsequently reviewed by the semantic checker. If a semantic misalignment is detected, the checker provides detailed feedback, marks the item with `inaccurate`, and the orchestrator assigns the statement to the semantic repairer to fix the issue. The repaired statement is then returned to the semantic checker for another round of validation. This iterative repair-and-check loop continues until the checker confirms that the formal statement is semantically aligned with the source statement. Statements are frozen before the proving stage unless the proving agent identifies a gap in the statement or its assumptions.

3.3 Theorem proving

The theorem proving stage replaces the remaining `sorry` placeholders with Lean proofs. The **proving orchestrator** coordinates four roles: **planner** (proof planning), **formalizer** (statement formalization), **prover** (proof generation) and **golfer** (proof golfing). The same DAG is used again, items in one depth layer can be proved concurrently, whereas a dependent layer waits until every item in the current layer has a compiling, frozen interface. This barrier preserves a stable context for parallel proving agents to make progress simultaneously.

For each item, the planner first analyzes the target `sorry` placeholders and reads the original natural language proof. It then develops a proof plan by decomposing the target into manageable subgoals, identifying potentially useful lemmas and tactics, and determining an appropriate proof strategy. Based on this plan, the formalizer states useful auxiliary lemmas and leaves their proofs as placeholders. These lemmas are kept in the same Lean file as the target theorem and are not added to the paper-level DAG. The prover is asked to construct the corresponding Lean proof and fill in the `sorry`. If the proof attempt fails, the prover returns diagnostic information and structured feedback to the orchestrator. The orchestrator then uses this feedback to request a revised proof plan and ask for another proof attempt. This plan–prove–feedback loop continues until all proof obligations have been discharged and the resulting Lean file contains no `sorry`. Only after repeated unsuccessful plan–prove iterations, when the planner explicitly identifies missing assumptions, may the proving

orchestrator add the corresponding assumptions to the statement. An item is considered proved only when it contains no `sorry`, `admit`, `axiom` or Lean errors.

Finally, the completed proof is passed to the subsequent proof-golfing stage, where it can be simplified and improved without changing its correctness. Our golf stage is based on Lean skills [1]. It also runs the `mathlib` linters and revises the code according to their diagnostics, including warnings about redundant arguments, deprecated declarations, unnecessary tactics, and noncanonical style.

3.4 Library integration

Library integration is performed after a batch of papers has been formalized. The integration agent first groups candidate definitions and theorems expressing the same mathematical content, and summarizes their occurrence and use across distinct papers. Frequently recurring candidates are prioritized for consolidation into shared interfaces, keeping `Foundation` focused on reusable content and limiting the accumulation of redundant or narrowly specialized declarations.

Approved candidates are consolidated into a common definition or theorem interface, reusing an existing `Foundation` API where appropriate. The affected paper projects are then updated and rebuilt against the shared interface. Paper-specific algorithms and principal convergence results remain in their respective projects. Figure 3 illustrates a candidate consolidation: two paper-local definitions of ε-enlargement can share a single `Foundation` interface.

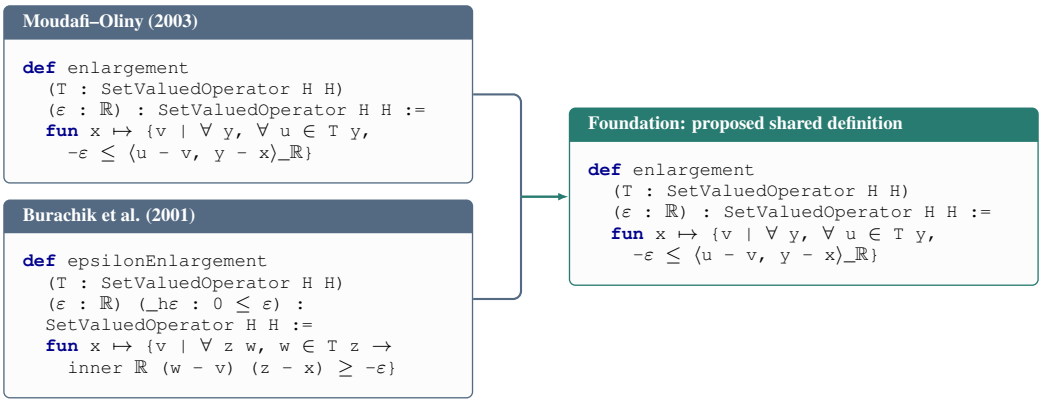


Figure 3: Proposed consolidation of ε-enlargement from Moudafi & Oliny (2003) and Burachik et al. (2001) into `Foundation`. $\varepsilon \geq 0$ is retained where needed in lemmas.

Paper-level completion is checked independently of batch library integration. After all DAG layers have reached a terminal state, Paper2Lean builds the complete collection of paper modules together with their import closures. A paper is marked complete only when every item is frozen, every module passes the final Lake build, and no proof placeholders remain.

4 Evaluation

In this section, we evaluate Paper2Lean on 20 research papers in mathematical optimization, focusing on inertial acceleration methods and dynamical-system approaches. We first describe the experimental setup and supporting libraries, and then report the scale and compilation status of the generated projects. We next evaluate the semantic fidelity of the formalized statements through expert review. Finally, a controlled comparison on three representative papers examines the performance and token consumption of two base models in statement generation and proof construction.

4.1 Setup

For document extraction, statement formalization, and proof construction, we use coding agents powered by Codex CLI, with `GPT-5.6-Terra` as the base model and the reason-

[1] `https://github.com/cameronfreer/lean4-skills`

ing effort set to `high`. All workers run as subagents. Throughout PAPER2LEAN, all the agents interact with Lean through Lean LSP MCP [2], which provides compiler feedback, goal-state inspection, and semantic search capabilities. All formalizations use the Lean toolchain `leanprover/lean4:v4.30.0-rc2`.

Research papers in optimization often rely on substantial background theory that is either unavailable in `mathlib` or represented through APIs that are not directly aligned with the notation and abstractions used in the source papers. To bridge this gap, we first use M2F (Wang et al., 2026) to formalize part of the book (Bauschke & Combettes, 2011) to construct a supporting `Foundation` library containing the core definitions and results required by the target papers. We also integrate some important theorems from Optlib (Li et al., 2025; 2026) and (Bai et al., 2026). PAPER2LEAN then formalizes each paper against the combination of this `Foundation` and `mathlib`. An overview of `Foundation` is provided in Appendix B.

Table 1 summarizes the resulting 20 papers including source-paper pages, number of generated Lean files, lines of Lean code, token usage, and source fidelity review. In total, all projects compile successfully and contain no `sorry` placeholders.

Number	Papers	Pages	Files	Loc	Token usage	Faithful
1	Abbas & Attouch (2015)	30	38	10819	275,269,684	37/38(97%)
2	Alvarez et al. (2002)	33	33	11752	131,267,936	27/33(82%)
3	Alvarez & Attouch (2001)	9	8	1428	11,135,901	8/8(100%)
4	Alvarez (2000)	18	22	7312	169,776,513	23/23(100%)
5	Alvarez (2003)	10	14	4315	84,065,469	14/14(100%)
6	Attouch et al. (2014)	25	42	6016	176,489,473	42/42(100%)
7	Adly & Attouch (2023)	43	25	4892	80,166,173	20/25(80%)
8	Ait Mansour et al. (2024)	26	41	7301	101,882,392	41/41(100%)
9	Attouch et al. (2024)	21	20	4444	84,315,239	20/21(95%)
10	Adly & Attouch (2020)	29	31	4965	89,058,313	28/32(88%)
11	Attouch et al. (2007)	21	25	8164	241,717,003	25/25(100%)
12	Baji & Cabot (2006)	23	32	15639	189,536,244	32/32(100%)
13	Beck & Teboulle (2009)	20	20	2210	68,458,794	20/20(100%)
14	Boź & Csetnek (2016)	17	24	6901	192,617,442	24/24(100%)
15	Burachik et al. (2001)	20	19	1684	40,900,232	19/19(100%)
16	Dong et al. (2017)	22	14	1943	39,369,773	14/14(100%)
17	Guler (1992)	16	35	2671	95,103,480	34/35(97%)
18	Lorenz & Pock (2015)	15	24	4332	58,099,932	18/24(75%)
19	Moudafi & Oliny (2003)	8	11	2015	47,541,502	11/11(100%)
20	Nesterov (1983)	5	16	2107	47,901,203	16/16(100%)

Table 1: Overview of 20 formalized papers. All generated projects compile without `sorry`. The Faithful column reports source fidelity of the direct output of PAPER2LEAN. Token usage includes statement formalization and proof construction, but excludes proof golfing.

4.2 SEMANTIC EVALUATION

Since the statement may be modified in proof stage, we therefore evaluate the statement by *source fidelity*. A statement in the output of PAPER2LEAN is called *faithful* when it preserves the mathematical meaning and scope of the source item *in the context of the full paper*. The review is done after every item has been proved and every project builds without `sorry`. This checkpoint mainly includes statement revisions authorized during proof construction. Restoring an assumption supported by the whole source context is acceptable; introducing an unsupported, non-redundant assumption that restricts the original claim is a semantic discrepancy, even if it enables a successful proof. Appendix D presents two examples: an acceptable revision that restores missing assumptions, and an unacceptable revision that assumes key facts in the proof instead of deriving them.

[2] `https://github.com/oOo0oOo/lean-lsp-mcp`

The human review is conducted by four experts in mathematical optimization with experience in large-scale Lean formalization. Each expert initially reviews five papers, then cross-audits the five papers assigned to another reviewer. Each of the 20 papers is therefore examined by two reviewers, and each expert reviews ten papers in total. Reviewers compare the Lean statements with the source papers for source fidelity, and the evaluation results can be found in Table 1.

Across the 20 papers, 473 of 494 items are faithful, giving an item-weighted source-fidelity rate of 95.7%. Thirteen papers have no detected discrepancy, while the remaining seven contain 21 unfaithful statements in total. This distribution shows that PAPER2LEAN preserves the source meaning for the large majority of statements. It also exposes a limitation: the 21 discrepancies are found in Lean-valid developments with no proof placeholders. In other words, successful proof construction establishes correctness relative to the generated statement, but does not by itself establish source fidelity.

4.3 MODEL COMPARISON

To estimate the sensitivity of PAPER2LEAN to the base model, we conduct a controlled study on three selected papers. All configurations use the same Lean, `mathlib` and `Foundation` version, tool access, and iteration limits. We report statement generation and proof construction separately.

4.3.1 STATEMENT GENERATION

Unlike *source fidelity*, we evaluate the quality of generated statement by *input alignment*. To evaluate this index, the statements generated in formalization stage (before proof construction) are assessed against the supplied JSON items. Theorem proof bodies may still contain `sorry`. A statement is called *aligned* when it faithfully expresses the natural language in the JSON item. Two reviewers examine all three test papers.

Using the same JSON input for each paper, we compare two base models—`GPT-5.6-Terra (high)` and `DeepSeek-V4.1-Flash (high)`, both run through the Codex CLI coding agent. We record the number of statements judged acceptable by the human review in Section 4, together with the token usage and the number of formalize-check rounds. Table 2 reports the results.

Papers	GPT-5.6-Terra			DeepSeek-V4.1-Flash		
	Aligned	Token	Round	Aligned	Token	Round
Moudafi & Oliny (2003)	11/11	13,391,172	1.55	11/11	9,236,271	1.09
Abbas & Attouch (2015)	37/38	60,160,380	1.84	37/38	56,656,127	1.03
Baji & Cabot (2006)	32/32	38,377,758	1.34	32/32	26,119,725	1.04

Table 2: Semantically aligned items, token usage (only statement formalization) and average formalize-check rounds in statement formalization stage under the same 15-round budget. The Aligned columns report input alignment before proof construction.

Both models perform well in statement generation, with token consumption on a comparable scale. `DeepSeek-V4.1-Flash` uses fewer tokens on each of the three papers, with a modest aggregate reduction of approximately 17.8% relative to `GPT-5.6-Terra`. These results suggest that both models can support the statement-generation stage, with `DeepSeek-V4.1-Flash` offering somewhat lower token usage in this comparison.

4.3.2 PROOF CONSTRUCTION

Using the same fixed statements, we compare proof construction in terms of proved items, token consumption, and plan-prove rounds, as shown in Table 3.

Both models complete all 81 proof-construction tasks across the three papers within the same iteration budget, including demanding items whose Lean developments exceed 1,000 lines. However, `DeepSeek-V4.1-Flash` consumes substantially more tokens: its total usage is approximately 3.41 times that of `GPT-5.6-Terra`, with per-paper ratios ranging from 1.29 to 4.31. Thus, both models can handle these proof tasks, but `GPT-5.6-Terra` achieves the same completion rate with substantially lower token usage under the evaluated workflow.

Papers	GPT-5.6-Terra			DeepSeek-V4.1-Flash		
	Proved	Token	Round	Proved	Token	Round
Moudafi & Oliny (2003)	11/11	34,150,330	1.25	11/11	44,088,265	1
Abbas & Attouch (2015)	38/38	215,109,304	1.6	38/38	926,283,566	1.47
Baji & Cabot (2006)	32/32	151,158,486	1.94	32/32	394,107,015	1.22

Table 3: Proved items, token usage, and average plan-prove rounds in proof construction under the same 10-round budget. Token usage includes only proof construction, excluding proof golfing.

4.4 OPTPAPER-20: DATASET AFTER HUMAN REFINEMENT

For all 20 papers formalized by PAPER2LEAN, we manually refine the generated code, primarily to improve the quality of the statements and remove redundant definitions or assumptions. These changes make the code more readable and reusable. Examples of human refinement can be found in the next section. We also translate the refined formal statements into natural language. The complete OPTPAPER-20 dataset contains the Lean files for all 20 papers, their corresponding natural language descriptions, and structured JSON files.

5 CASE STUDIES AND DISCUSSIONS

In this section, we examine two cases of PAPER2LEAN output. The first is semantically faithful in the context of the full paper but can be refined to make downstream theorem statements more concise and readable. The second passes the agent's semantic check and Lean proof verification yet is judged unfaithful because it omits the source claim and its contextual dependencies.

5.1 CASE 1: FAITHFUL, BUT REQUIRING REFINEMENT.

We judge the original trajectory definition for the Hessian-driven inertial equation (DIN) in Figure 4 *semantically faithful* in the context of Alvarez et al. (2002): downstream assumptions identify the supplied gradient and Hessian maps with those of the potential, while `HasDerivAt` conditions identify velocity and acceleration. However, these auxiliary parameters make the interface cumbersome to reuse. The refinement expresses derivatives directly through `mathlib`'s `gradient`, `fderiv`, and `deriv`, and restricts the equation to $t \geq 0$, matching its use in subsequent theorems. Appendix C presents the corresponding downstream assumptions and weak convergence theorem before and after refinement. Removing auxiliary map parameters and repeated compatibility hypotheses makes the downstream theorem statements less verbose and easier to read.

(a) Before: faithful statement

```
def IsDINTrajectory (H : Type*)
  [NormedAddCommGroup H]
  [InnerProductSpace ℝ H]
  (gradΦ : H → H)
  (hessΦ : H → H →L[ℝ] H)
  (α β : ℝ) (x v a : ℝ → H) : Prop :=
  0 < α ∧ 0 < β ∧
  (∀ t, HasDerivAt x (v t) t) ∧
  (∀ t, HasDerivAt v (a t) t) ∧
  ∀ t, a t + α • v t +
  β • hessΦ (x t) (v t) +
  gradΦ (x t) = 0
```

(b) After: refined statement

```
def IsDINTrajectory (H : Type*)
  [NormedAddCommGroup H]
  [InnerProductSpace ℝ H]
  [CompleteSpace H] (Φ : H → ℝ)
  (α β : ℝ) (x : ℝ → H) : Prop :=
  0 < α ∧ 0 < β ∧ ∀ t, 0 ≤ t →
  deriv (deriv x) t + α • deriv x t +
  β • (fderiv ℝ (gradient Φ) (x t))
  (deriv x t) + gradient Φ (x t) = 0
```

Figure 4: DIN trajectory in Alvarez et al. (2002): (a) original statement; (b) refined statement.

This refinement improves usability but requires care with assumptions. The predicates are not unconditionally equivalent because their time domains differ, and downstream results must retain the required regularity of Φ and twice differentiability of the trajectory: `deriv` alone does not assert differentiability. The case therefore motivates refining a semantically faithful statement for reuse while preserving its mathematical obligations.

5.2 CASE 2: UNFAITHFUL DESPITE PASSING AGENT CHECKS AND PROOF.

The formalization of Remark 2.19 in Abbas & Attouch (2015) (Figure 5) passed the formalization agent's semantic check and was successfully proved in Lean, but we judge it *not semantically faithful*. The source claims strong convergence of the system's trajectory under the Kadec–Klee property. Under the standing assumptions, including $S \neq \emptyset$, Theorem 2.8 and Proposition 2.16 provide weak convergence to $x_\infty \in S$ and convergence of $\Phi(x(t))$ to $\Phi(x_\infty)$; applying the property along sequences $t_k \to +\infty$ yields $\|x(t) - x_\infty\| \to 0$.

(a) Natural-language source: Remark 2.19

"By a similar argument, strong convergence of $x(\cdot)$ holds under the Kadec-Klee property: whenever (x_k) converges weakly to x, and $(\Phi(x_k))$ converges to $\Phi(x)$, then (x_k) converges strongly to x."

(b) Generated Lean statement and proof

```
theorem kadec_klee_property {H : Type*}
    [NormedAddCommGroup H] [NormedSpace ℝ H] [InnerProductSpace ℝ H]
    (Phi : H → Set.Ioi (⊥ : EReal))
    (_hPhi : Phi ∈ ERealFunction.gammaZero H) :
    (hKadecKlee : ∀ (xSeq : ℕ → H) (x : H),
      Tendsto (fun k ↦ toWeakSpace ℝ H (xSeq k)) atTop (nhds (toWeakSpace ℝ H x)) →
      Tendsto (fun k ↦ (Phi (xSeq k) : EReal)) atTop (nhds (Phi x : EReal)) →
      Tendsto (fun k ↦ ‖xSeq k - x‖) atTop (nhds 0)) → ∀ (xSeq : ℕ → H) (x : H),
      Tendsto (fun k ↦ toWeakSpace ℝ H (xSeq k)) atTop (nhds (toWeakSpace ℝ H x)) →
      Tendsto (fun k ↦ (Phi (xSeq k) : EReal)) atTop (nhds (Phi x : EReal)) →
      Tendsto (fun k ↦ ‖xSeq k - x‖) atTop (nhds 0) := id
```

Figure 5: Remark 2.19 of Abbas & Attouch (2015) and its generated formalization, which passed automatic semantic checking and Lean proof verification.

The hypothesis `hKadecKlee` and the conclusion express the same property, so `id` proves the statement. The claim about strong convergence of the source paper's trajectory is absent. The assumption itself is legitimate, but omitting its consequence for the trajectory loses the source claim. This illustrates how an agent can miss the context behind "by a similar argument" even when semantic checking and proof construction both succeed. Review must therefore check the intended conclusion against the surrounding paper and its dependencies.

6 CONCLUSION

We introduced PAPER2LEAN, an agentic framework for converting mathematical research papers into integrated Lean projects through dependency-aware extraction, formalization, semantic checking, theorem proving, and library integration. Its central representation is a dependency DAG that supports staged processing, compilation-gated validation, and the reuse of verified interfaces across a complete paper. Evaluated on 20 optimization papers with `mathlib` and a shared supporting library, the framework generated projects that compile without proof placeholders. The resulting OPTPAPER-20 corpus contains 494 Lean files and 110,910 lines of code, together with the natural language and structured JSON files.

Our audit also shows that Lean-checked compilation and faithful formalization measure different properties. Before manual correction, 473 of 494 reviewed declarations were judged faithful; all identified discrepancies were repaired in the released corpus. This finding supports paper-level formalization as a practical direction while showing that semantic review remains necessary for detecting omitted assumptions, altered claims, and lost contextual dependencies. Future work will focus on automated semantic checking (Liu et al., 2026d; 2025), better recovery of implicit assumptions, lower token costs, broader mathematical domains, and more systematic comparisons with alternative agent architectures and formalization pipelines.

AI USE STATEMENT

In this work, we used generative AI tools throughout PAPER2LEAN to extract mathematical items and dependencies; formulate Lean definitions, theorem statements, and intermediate claims; check and repair semantic misalignments; plan, prove in Lean, and golf formal proofs. We did not use generative AI to replace the human expert audit of OPTPAPER-20, determine its discrepancy classifications, or interpret the reported evaluation results. Additionally, we used generative AI to improve the manuscript's readability. We have reviewed all AI-assisted work. We reviewed the resulting LLM generated formal statements and code. We take responsibility for the final content, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work formalizes published mathematical research papers and involves no human subjects, personal or sensitive data, or high-stakes deployment. We are not aware of additional ethical concerns specific to this study.

REPRODUCIBILITY STATEMENT

The framework and experiment setup are described in Sections 3 and 4; the released artifacts include the pipeline, configurations, pinned Lean environment, generated and expert-corrected projects. Starting from a paper PDF, readers can run PAPER2LEAN with Codex under the reported configuration to regenerate a complete formalization with comparable token consumption. All the code and dataset can be found in `https://anonymous.4open.science/r/Paper2Lean-5A6D/`.

REFERENCES

Boushra Abbas and Hedy Attouch. Dynamical systems and forward–backward algorithms associated with the sum of a convex subdifferential and a monotone cocoercive operator. *Optimization*, 64(10):2223–2252, 2015.

Samir Adly and Hedy Attouch. Finite convergence of proximal-gradient inertial algorithms combining dry friction with hessian-driven damping. *SIAM Journal on Optimization*, 30(3):2134–2162, 2020.

Samir Adly and Hedy Attouch. Accelerated dynamics with dry friction via time scaling and averaging of doubly nonlinear evolution equations. *Nonlinear Analysis: Hybrid Systems*, 50:101402, 2023.

Mohamed Ait Mansour, Hedy Attouch, and Zakaria Mazgouri. Second order dynamical systems governed by maximally monotone operators: Quantitative stability. *Evol. Equ. Control Theory*, 13:510–535, 2024.

Felipe Alvarez. On the minimizing property of a second order dissipative system in hilbert spaces. *SIAM Journal on Control and Optimization*, 38(4):1102–1119, 2000.

Felipe Alvarez. Weak convergence of a relaxed and inertial hybrid projection-proximal point algorithm for maximal monotone operators in hilbert space. *SIAM J. on Optimization*, 14(3):773–782, March 2003. ISSN 1052-6234.

Felipe Alvarez and Hedy Attouch. An inertial proximal method for maximal monotone operators via discretization of a nonlinear oscillator with damping. *Set-Valued Analysis*, 9(1):3–11, 2001.

Felipe Alvarez, Hedy Attouch, Jérôme Bolte, and Patrick Redont. A second-order gradient-like dissipative dynamical system with hessian-driven damping.: Application to optimization and mechanics. *Journal de mathématiques pures et appliquées*, 81(8):747–779, 2002.

Anthropic. Fermat's last theorem in Lean 4. `https://github.com/anthropics/fermats-last-theorem`, September 2026. Complete machine-checked formalization; accessed September 22, 2026.

H. Attouch, P. Redont, and A. Soubeyran. A new class of alternating proximal minimization algorithms with costs-to-move. *SIAM Journal on Optimization*, 18(3):1061–1081, 2007.

Hédy Attouch, Juan Peypouquet, and Patrick Redont. A dynamical approach to an inertial forward-backward algorithm for convex minimization. *SIAM Journal on Optimization*, 24(1):232–256, 2014.

Hedy Attouch, Zaki Chbani, and Hassan Riahi. Fast convex optimization via a third-order in time evolution equation: Toges-v an improved version of toges. *Optimization*, 73(3):575–595, 2024.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

Yifan Bai, Yantao Li, Jian Yu, and Jingwei Liang. Formalization of two fixed-point algorithms in hilbert spaces. *arXiv preprint arXiv:2602.17064*, 2026.

Bruno Baji and Alexandre Cabot. An inertial proximal algorithm with dry friction: Finite convergence results. *Set-Valued Analysis*, 14:1–23, 01 2006.

Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq proof assistant reference manual: Version 6.1*. PhD thesis, Inria, 1997.

H. Bauschke and P. L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, March 2009.

Radu Ioan Boź and Ernö Robert Csetnek. An inertial tseng's type proximal algorithm for nonsmooth and nonconvex optimization problems. *J. Optim. Theory Appl.*, 171(2):600–616, November 2016. ISSN 0022-3239.

Regina S Burachik, Susana Scheimberg, and Benar F Svaiter. Robustness of the hybrid extragradient proximal-point algorithm. *Journal of Optimization Theory and Applications*, 111(1):117–136, 2001.

Rafael Cabral, Tuan Manh, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Shen Xin. Proofflow: A dependency graph approach to faithful proof autoformalization. In *International Conference on Learning Representations*, volume 2026, pp. 58459–58483, 2026.

Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pp. 378–388. Springer, 2015.

DeepSeek-AI. Deepseek harness: Everything is a plugin. `https://github.com/deepseek-ai/deepseek-harness`, 2026.

Qiao-Li Dong, Dan Jiang, Prasit Cholamjiak, and Yekini Shehu. A strong convergence result involving an inertial forward–backward algorithm for monotone inclusions. *Journal of Fixed Point Theory and Applications*, 19, 12 2017.

Osman Guler. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2:649–664, 01 1992.

Wang Hanyu, Ruohan Xie, Wang Yutong, Guoxiong Gao, and Bin Dong. Aria: An agent for retrieval and iterative auto-formalization via dependency graph. In *International Conference on Learning Representations*, volume 2026, pp. 11690–11722, 2026.

Haocheng Ju, Leheng Chen, Peihao Wu, Bryan Dai, and Bin Dong. Matlas: A semantic search engine for mathematics. *arXiv preprint arXiv:2604.17484*, 2026a.

Haocheng Ju, Guoxiong Gao, Jiedong Jiang, Bin Wu, Zeming Sun, Shurui Liu, Leheng Chen, Yutong Wang, Yuefeng Wang, Zichen Wang, et al. Automated conjecture resolution with formal verification. *arXiv preprint arXiv:2604.03789*, 2026b.

Chenyi Li, Ziyu Wang, Wanyi He, Yuxuan Wu, Shengyang Xu, and Zaiwen Wen. Formalization of convergence rates of four first-order algorithms for convex optimization: C. li et al. *Journal of Automated Reasoning*, 69(4):28, 2025.

Chenyi Li, Zichen Wang, Yifan Bai, Yunxi Duan, Yuqing Gao, Pengfei Hao, and Zaiwen Wen. Formalization of algorithms for optimization with block structures. *Science China Mathematics*, pp. 1–18, 2026.

Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, et al. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. *arXiv preprint arXiv:2601.14027*, 2026a.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *International Conference on Learning Representations*, volume 2025, pp. 86205–86233, 2025.

Xiaoyang Liu, Kangjie Bao, Jiashuo Zhang, Yunqi Liu, Yu Chen, Yuntian Liu, Yang Jiao, and Tao Luo. Atlas: Autoformalizing theorems through lifting, augmentation, and synthesis of data. *Advances in Neural Information Processing Systems*, 38:56290–56326, 2026b.

Xiaoyang Liu, Zineng Dong, Yifan Bai, Yantao Li, Yuntian Liu, and Tao Luo. Decompose, structure, and repair: A neuro-symbolic framework for autoformalization via operator trees. *arXiv preprint arXiv:2604.19000*, 2026c.

Xiaoyang Liu, Tao Zhu, Zineng Dong, Yuntian Liu, Liu ZhaoXuan, Yu Chen, Tao Luo, et al. Assess: A semantic and structural evaluation framework for statement similarity. In *International Conference on Learning Representations*, volume 2026, pp. 143709–143734, 2026d.

Dirk A Lorenz and Thomas Pock. An inertial forward-backward algorithm for monotone inclusions. *Journal of Mathematical Imaging and Vision*, 51(2):311–325, 2015.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP '20. ACM, January 2020.

A. Moudafi and M. Oliny. Convergence of a splitting inertial proximal method for monotone operators. *Journal of Computational and Applied Mathematics*, 155(2):447–454, 2003.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, pp. 543, 1983.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

George Tsoukalas, Anton Kovsharov, Sergey Shirobokov, Anja Surina, Moritz Firsching, Gergely Bérczi, Francisco J. R. Ruiz, Arun Suggala, Adam Zsolt Wagner, Eric Wieser, Lei Yu, Aja Huang, Miklós Z. Horváth, Andrew Ferrauiolo, Henryk Michalewski, Edward Lockhart, Codrut Grosu, Thomas Hubert, Matej Balog, Pushmeet Kohli, and Swarat Chaudhuri. Advancing mathematics research with AI-driven formal proof search. *arXiv preprint arXiv:2605.22763*, 2026.

Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, et al. Mineru: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*, 2024.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Zichen Wang, Wanli Ma, Zhenyu Ming, Gong Zhang, Kun Yuan, and Zaiwen Wen. M2f: Automated formalization of mathematical literature at scale. *arXiv preprint arXiv:2602.17016*, 2026.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A Example Intermediate Artifact

Extraction output. After document extraction, Paper2Lean stores the mathematical content of each source paper in a structured JSON artifact, which mainly records paper-level metadata and a list of mathematical items. Each item has a stable identifier and type, a human-readable name, the extracted mathematical statement, a natural language proof (optional), and explicit `depends_on` edges. These edges form the initial dependency graph used to schedule downstream statement formalization and proof construction. Listing 1 is a faithful excerpt from the extraction output for Beck & Teboulle (2009). It follows a representative chain from the optimization model and proximal map, through an optimality lemma and the FISTA algorithm, to the paper's main convergence theorem. All displayed values are reproduced from the generated artifact. For readability, the long `proof` field of the final theorem is omitted from the selected records.


```
{
  "metadata": {
    "title": "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse
  Problems",
    "authors": [
      "Amir Beck",
      "Marc Teboulle"
    ],
    "year": 2009,
    "source_type": "markdown",
    "venue": "SIAM Journal on Imaging Sciences, Vol. 2, No. 1, pp. 183-202"
  },
  "items": [
    {
      "id": "def_5",
      "type": "definition",
      "name": "General convex optimization model (Problem P)",
      "statement": "The general nonsmooth convex optimization model is $$(\\mathrm{P})\\
  quad \\min\\{F(x) \\equiv f(x) + g(x) : x \\in \\mathbb{R}^n\\},$$ where:\\n- $g : \\
  mathbb{R}^n \\to \\mathbb{R}$ is a continuous convex function which is possibly
  nonsmooth.\\n- $f : \\mathbb{R}^n \\to \\mathbb{R}$ is a smooth convex function of type
   $C^{1,1}$, i.e., continuously differentiable with Lipschitz continuous gradient $L(f)$
  .\\n- Problem (P) is solvable, i.e., $X^* := \\arg\\min F \\ne \\emptyset$, and for $x
  ^* \\in X^*$ we set $F_* := F(x^*)$.",
      "proof": null,
      "depends_on": ["def_3"]
    },
    {
      "id": "def_7",
      "type": "definition",
      "name": "Proximal operator $p_L(y)$",
      "statement": "For any $L > 0$ and $y \\in \\mathbb{R}^n$, the proximal operator (or
   prox-map) is defined as the unique minimizer $$p_L(y) := \\arg\\min\\{Q_L(x,y) : x \\
  in \\mathbb{R}^n\\}.$$ Equivalently, ignoring constant terms in $y$, $$p_L(y) = \\arg\\
  min_x \\left\\{ g(x) + \\frac{L}{2}\\left\\|x - \\left(y - \\frac{1}{L}\\nabla f(y)\\
  right)\\right\\|^2 \\right\\}.$$",
      "proof": null,
      "depends_on": ["def_6"]
    },
    {
      "id": "lem_2",
```

```
      "type": "lemma",
      "name": "Lemma 2.2: Optimality characterization of the proximal operator",
      "statement": "For any $y \\in \\mathbb{R}^n$, one has $z = p_L(y)$ if and only if
  there exists $\\gamma(y) \\in \\partial g(z)$, the subdifferential of $g(\\cdot)$, such
   that $$\\nabla f(y) + L(z-y) + \\gamma(y) = 0.$$",
      "proof": "The proof is immediate from optimality conditions for the strongly convex
   problem (2.6). Since $Q_L(\\cdot, y)$ is strongly convex with modulus $L$, the first-
  order necessary and sufficient optimality condition is $0 \\in \\partial(Q_L(\\cdot, y)
  )(z) = \\nabla f(y) + L(z-y) + \\partial g(z)$.",
      "depends_on": [
        "def_4",
        "def_7"
      ]
    },
    {
      "id": "algo_1",
      "type": "algorithm",
      "name": "FISTA with constant stepsize",
      "statement": "FISTA with constant stepsize: the sequences satisfy the proximal
  update and the momentum recurrences of equations (4.1)--(4.3).",
      "proof": null,
      "depends_on": ["def_5", "def_7"]
    },
    {
      "id": "thm_2",
      "type": "theorem",
      "name": "Theorem 4.4: Global convergence rate of FISTA -- $O(1/k^2)$",
      "statement": "Let $\\{x_k\\}$, $\\{y_k\\}$ be generated by FISTA with either
  constant or backtracking stepsize rule. Then for any $k \\ge 1$, $$F(x_k) - F(x^*) \\le
   \\frac{2\\alpha L(f)\\|x_0 - x^*\\|^2}{(k+1)^2} \\quad \\forall x^* \\in X^*,$$ where
  $\\alpha = 1$ for the constant stepsize setting and $\\alpha = \\eta$ for the
  backtracking stepsize setting.",
      "depends_on": [
        "def_5",
        "def_7",
        "lem_3",
        "lem_4",
        "lem_5",
        "lem_6"
      ]
    }
  ]
}
```


Listing 1: Excerpt from `paper_expanded.json` for Beck & Teboulle (2009), showing extracted definitions, a lemma, an algorithm, a theorem, and their dependency edges.

The excerpt illustrates how heterogeneous paper content is represented in one uniform schema. The definitions introduce the composite objective and its proximal map; `lem_2` records the first-order optimality characterization of that map; `algo_1` identifies the FISTA update scheme; and `thm_2` states the accelerated $O(1/k^2)$ objective-value rate. The explicit dependency lists retain the semantic prerequisites for each item. In particular, the theorem depends on the model and proximal-map definitions as well as the four lemmas used in its proof, allowing downstream agents to formalize and verify these declarations in dependency order.

B Shared Foundation Library

Purpose and scope. The paper projects use a shared Lean library, called `Foundation`, for reusable interfaces and supporting results in convex analysis and optimization. The library addresses mathematical components that are only partially available in `mathlib` or whose `mathlib` APIs do not match the notation used in the source papers. It is a supporting library rather than a monolithic import: each paper imports only the modules required by its own definitions and proofs.

Scale and organization. The released version of `Foundation` contains 61 Lean modules, 48,005 lines of source code (37,806 non-comment and nonblank lines), and approximately 1,257 top-level declarations (definitions, lemmas, theorems, structures, classes, instances, and abbreviations). The modules are organized into the 13 thematic namespaces shown in Table 4. The largest components are convex analysis, fixed-point theory, differentiation and ordinary differential equations, and Hilbert space analysis.

Namespace	Modules	Lines	Code lines	Main contents
Analysis	6	3,277	2,574	Extended-real functions, integrals, sequences, semicontinuity, limiting subdifferentials
Convex	17	13,416	10,588	Convexity, Fenchel–Rockafellar duality, subdifferentials, proximal/Moreau operators
Differentiation	4	6,699	5,816	Gâteaux and Lipschitz differentiation, Moreau gradients, ODEs
FixedPoint	6	6,462	4,964	Nonexpansive, averaged, quasi-nonexpansive and inertial maps
Hilbert	5	4,919	3,578	Hilbert geometry, orthogonality, weak topology and convergence
Metric	1	577	419	Metric fixed-point results
Operator	3	1,323	934	Linear operators, distances and bifunctions
Optimization	3	697	531	Argmin, descent structures, and Kurdyka–Lojasiewicz results
Order	1	254	142	Extended-real order structures
Projection	1	2,316	1,657	Best approximation and projection results
Separation	1	1,018	766	Separating hyperplanes
SetValued	10	5,562	4,823	Monotone/maximal-monotone operators, enlargements, product normal cones, resolvents, Minty theory
Topology	3	1,485	1,014	Topological and sequence-space utilities
Total	**61**	**48,005**	**37,806**	

Table 4: Organization of the shared `Foundation` library. "Lines" counts all source lines, including comments and blank lines; "Code lines" excludes comments and blank lines.

Dependency accounting. Each paper project is compiled against the same pinned versions of Lean, `mathlib`, and `Foundation`. We distinguish three dependency layers: `mathlib` provides general mathematical infrastructure, `Foundation` provides reusable optimization and convex analysis interfaces, and each paper project contains paper-specific declarations. The statistics below count explicit imports of modules whose names begin with `Foundation`; they therefore measure library-level reuse. Declaration-level dependencies can be obtained from Lean's environment and are kept as part of the released provenance data.

Across the 20 papers, 38 of the 61 `Foundation` modules are directly imported by at least one project. The number of directly imported modules per paper ranges from 0 to 14 (mean 7.90). The most widely reused interfaces are `Hilbert.WeakConvergence` (14 papers), `Analysis.ExtendedRealFunctions` (12), `Convex.Subdifferential` (10), `SetValued.Monotone` and `SetValued.Resolvent` (8 each), and `FixedPoint.Nonexpansive`, `Hilbert.WeakTopology`, and `SetValued.Basic` (7 each). Table 5 gives the complete module-level usage summary, and Table 6 gives the paper-level import summary.

How the papers use the library. The dynamical-system papers primarily use Hilbert-space geometry, weak topology, weak convergence, differentiation, and ODE interfaces. Proximal and forward–backward papers use convexity, subdifferentials, proximal operators, resolvents, and monotone-operator modules. Fixed-point and projection papers reuse nonexpansive, averaged, fixed-point, and best-approximation interfaces. Extended-real functions, semicontinuity, argmin, and coercivity form the common variational-analysis layer. Each paper imports a different subset according to its assumptions and proof structure.

Content boundary. `Foundation` is a shared supporting library, rather than a collection of the 20 dataset papers' results. It contains generic mathematical interfaces, including parameterized inertial or fixed-point schemas and general supporting lemmas, but excludes the paper-specific algorithm update rules, parameter regimes, and principal convergence theorems of the 20 dataset papers. Thus, `Foundation` may provide a general result about a nonexpansive map or a resolvent, while a concrete FISTA, inertial forward–backward, proximal, or dynamical-system algorithm and its main convergence statement are formalized in the relevant paper project. The import table measures module-level reuse; it is not by itself a theorem-level proof that no paper-specific declaration occurs in `Foundation`.

Module	Papers	Module	Papers
Hilbert.WeakConvergence	14	Analysis.ExtendedRealFunctions	12
Convex.Subdifferential	10	SetValued.Monotone	8
SetValued.Resolvent	8	FixedPoint.Nonexpansive	7
Hilbert.WeakTopology	7	SetValued.Basic	7
Convex.WeakClosed	6	Projection.BestApproximation	6
Convex.ProximalOperator	5	Differentiation.Gateaux	5
SetValued.Minty	5	Convex.StronglyConvex	4
Differentiation.ODE	4	Hilbert.Basic	4
Optimization.Argmin	4	SetValued.MaximalMonotone	4
Differentiation.MoreauGradient	3	Operator.Linear	3
SetValued.Enlargement	3	SetValued.MaximalMonotoneSum	3
Analysis.Semicontinuity	2	Convex.ArgminExistence	2
Convex.Basic	2	Convex.Conjugate	2
Convex.MoreauEnvelope	2	Convex.NormalCone	2
FixedPoint.Algorithms	2	Metric.FixedPoint	2
SetValued.Yosida	2	Topology.Sequences	2
Analysis.LimitingSubdifferential	1	Convex.Coercive	1
FixedPoint.Averaged	1	Operator.Distance	1
Optimization.KurdykaLojasiewicz	1	Separation.Hyperplanes	1

Table 5: Number of OPTPAPER-20 projects that directly import each `Foundation` module. Modules with zero direct imports are omitted.

Paper	Lean files	LOC	`Foundation` modules
Abbas & Attouch (2015)	38	10,819	10
Alvarez et al. (2002)	33	11,752	12
Alvarez & Attouch (2001)	8	1,428	6
Alvarez (2000)	22	7,312	9
Alvarez (2003)	14	4,315	9
Attouch et al. (2014)	42	6,016	9
Adly & Attouch (2023)	25	4,892	6
Ait Mansour et al. (2024)	41	7,301	9
Attouch et al. (2024)	20	4,444	10
Adly & Attouch (2020)	31	4,965	7
Attouch et al. (2007)	25	8,164	4
Baji & Cabot (2006)	32	15,639	14
Beck & Teboulle (2009)	20	2,210	0
Boź & Csetnek (2016)	24	6,901	8
Burachik et al. (2001)	19	1,684	12
Dong et al. (2017)	14	1,943	9
Guler (1992)	35	2,671	4
Lorenz & Pock (2015)	24	4,332	9
Moudafi & Oliny (2003)	11	2,015	7
Nesterov (1983)	16	2,107	4

Table 6: Direct `Foundation` imports in the 20 paper projects. LOC includes comments and blank lines; the final column counts distinct `Foundation` modules imported anywhere under the paper directory.

Validation and versioning. `Foundation` is compiled as an independent Lean library under the same toolchain and `mathlib` revision used for the paper projects. The released snapshot is identified by its repository revision and is frozen for the reported experiments. Every paper project is rebuilt from a clean environment after integration. The final audit checks for compilation errors, proof placeholders ("`sorry`" and "`admit`"), and unauthorized axioms. Results introduced after the frozen snapshot, if any, are reported as paper-local extensions and are not counted as part of the shared `Foundation` statistics.

C DOWNSTREAM THEOREM INTERFACE REFINEMENT

The reformulation in Figure 4 also simplifies the downstream convergence theorem. The original theorem carries the independently supplied gradient, Hessian, velocity, and acceleration maps, together with hypotheses identifying these maps with the corresponding derivatives. The revised theorem takes the potential and trajectory directly: the derivative APIs are used in the trajectory definition, while the required regularity of the trajectory is stated explicitly. The following listing shows the two interfaces for the DIN weak-convergence theorem from Alvarez et al. (2002).

(a) Before: expanded interface

```
structure AssumptionH (H : Type*)
    [NormedAddCommGroup H] [InnerProductSpace ℝ H
     ]
    [CompleteSpace H] (Φ : H → ℝ)
    (gradΦ : H → H) (hessΦ : H → H →L[ℝ] H) :
      Prop where
  bounded_below : ∃ m : ℝ, ∀ x, m ≤ Φ x
  twice_continuously_differentiable : ContDiff ℝ
     2 Φ
  hessian_is_derivative : ∀ x, HasFDerivAt gradΦ
      (hessΦ x) x
  hessian_lipschitz_on_bounded :
    ∀ R : ℝ, 0 < R → ∃ L : ℝ, 0 < L → ∀ x y,
      ‖x‖ ≤ R → ‖y‖ ≤ R →
      ‖hessΦ x - hessΦ y‖ ≤ L * ‖x - y‖

theorem din_weakly_converges_to_argmin
    {H : Type*} [NormedAddCommGroup H]
    [InnerProductSpace ℝ H] [CompleteSpace H]
    (Φ : H → ℝ) (gradΦ : H → H)
    (hessΦ : H → H →L[ℝ] H) (α β : ℝ)
    (x v acc : ℝ → H) (hΦ : AssumptionH H Φ
     gradΦ hessΦ)
    (hgradΦ : ∀ q, HasFDerivAt Φ (innerSL ℝ (
     gradΦ q)) q)
    (hhessΦ : ∀ R : ℝ, 0 < R → ∃ L : ℝ, 0 < L
     ∧ ∀ p q,
      ‖p‖ ≤ R → ‖q‖ ≤ R →
      ‖hessΦ p - hessΦ q‖ ≤ L * ‖p - q‖)
    (hconvex : ConvexOn ℝ Set.univ Φ)
    (hmin : {z | ∀ y, Φ z ≤ Φ y}.Nonempty)
    (hsol : IsDINTrajectory H gradΦ hessΦ α β x
      v acc) :
    (∀ z, (∀ y, Φ z ≤ Φ y) → ∃ d : ℝ,
      Tendsto (fun t => ‖x t - z‖) atTop (nhds d)
     ) ∧
      ∃ z, (∀ y, Φ z ≤ Φ y) ∧
        Tendsto (fun t => toWeakSpace ℝ H (x t))
      atTop
          (nhds (toWeakSpace ℝ H z)) := by
  ...
```

(b) After: compact interface

```
structure AssumptionH (H : Type*)
    [NormedAddCommGroup H] [InnerProductSpace ℝ H
     ]
    [CompleteSpace H] (Φ : H → ℝ) : Prop where
  bounded_below : ∃ m : ℝ, ∀ x, m ≤ Φ x
  twice_continuously_differentiable : ContDiff ℝ
     2 Φ
  hessian_lipschitz_on_bounded :
    ∀ R : ℝ, 0 < R → ∃ L : ℝ, 0 < L ∧ ∀ x y,
      ‖x‖ ≤ R → ‖y‖ ≤ R →
      ‖fderiv ℝ (gradient Φ) x -
        fderiv ℝ (gradient Φ) y‖ ≤ L * ‖x - y‖

theorem din_weakly_converges_to_argmin
    {H : Type*} [NormedAddCommGroup H]
    [InnerProductSpace ℝ H] [CompleteSpace H]
    (Φ : H → ℝ) (α β : ℝ) (x : ℝ → H)
    (hΦ : AssumptionH H Φ) (hx : ContDiff ℝ 2 x)
    (hconvex : ConvexOn ℝ Set.univ Φ)
    (hmin : (Optimization.argminSet Φ).Nonempty)
    (hsol : IsDINTrajectory H Φ α β x) :
    (∀ z, z ∈ Optimization.argminSet Φ → ∃ d :
     ℝ,
      Tendsto (fun t => ‖x t - z‖) atTop (nhds d)
     ) ∧
      ∃ z, z ∈ Optimization.argminSet Φ ∧
        Tendsto (fun t => toWeakSpace ℝ H (x t))
      atTop
          (nhds (toWeakSpace ℝ H z)) := by
  ...
```

Figure 6: Downstream interface comparison for the DIN weak convergence theorem. The revised statement removes repeated map-identification hypotheses, uses the shared `argminSet` abstraction, and makes the trajectory regularity assumption explicit. Proof bodies are omitted.

The main benefit is interface locality. In the original version, every theorem that uses the trajectory must thread the auxiliary maps and separately restate their compatibility with Φ. In the revised version these relationships are encoded once in the definitions used by the trajectory, while `ContDiff` on x records the regularity needed to interpret the second derivative. The theorem statement is consequently shorter and its hypotheses are closer to the mathematical objects they constrain. This refactoring does not remove mathematical obligations: derivative regularity and the identification of the gradient and Hessian are still required, but they are represented through shared APIs and reusable assumptions rather than repeated paper-local parameters.

D Statement Revisions During Proof Construction

D.1 Case 1: Recovering Assumptions Used in the Source Proof

During proof construction for Theorem 2.1 of Moudafi & Oliny (2003), the statement was augmented with `hS_nonempty` and `h_norm_diff_tendsto` (Figure 7). We regard these additions as recovering assumptions from the source proof, based on the following evidence.

Existence of a solution. The proof begins by fixing a point in $S = (A+B)^{-1}(0)$ (source p. 451). The hypothesis `hS_nonempty` makes this implicit existence assumption explicit and supports the later application of Opial's lemma; it does not assume convergence of the iterates.

Asymptotic regularity. The proof later invokes "by hypothesis" the condition $\|x_k - x_{k-1}\| \to 0$ (source p. 453), although this condition is not listed in the theorem statement. The hypothesis

h_norm_diff_tendsto expresses the same condition with a shifted index. Our assessment follows the theorem together with its proof: the revision makes this dependency explicit, but does not establish that it follows from the theorem's other listed assumptions. This distinguishes recovery of a source-supported condition from adding an unsupported restriction merely to complete a proof.

Theorem 2.1: statement after assumption recovery

```
theorem convergence_of_splitting_inertial_proximal
    {A : SetValuedOperator H H} {B : H → H}
    (hA_max : IsMaximalMonotone A)
    (hB_max : IsMaximalMonotone (svEmbed B))
    {β : ℝ} (hB_coco : IsCocoerciveWRT A B β)
    (hB_lipschitz : ∀ x y : H, ‖B x - B y‖ ≤ (1 / β) * ‖x - y‖)
    (x : ℕ → H) (θ lam ε : ℕ → ℝ) (y : ℕ → H)
    (hy : ∀ k, 1 ≤ k → y k = x k + θ k • (x k - x (k - 1)))
    (h_alg : ∀ k, 1 ≤ k →
      (0 : H) ∈ (lam k • enlargement A (ε k) (x (k + 1))) +
        {x (k + 1) - y k} + (lam k • svEmbed B (x k)))
    (h_delta : ∃ (δ : ℝ) (lamUnder : ℝ), 0 < lamUnder ∧ 0 < δ ∧
      ∀ k, 1 ≤ k → lamUnder ≤ lam k ∧ lam k ≤ 2 * β - δ)
    (h_theta : ∃ (θmax : ℝ), 0 ≤ θmax ∧ θmax < 1 ∧
      ∀ k, 1 ≤ k → 0 ≤ θ k ∧ θ k ≤ θmax)
    (h_eps_sum : Summable ε) (h_eps_pos : ∀ k, ε k ≥ 0)
    (h_summable : Summable fun k : ℕ ↦
      θ (k + 1) * ‖x (k + 1) - x k‖ ^ 2)
    (hS_nonempty : ((A + svEmbed B).zeros).Nonempty)
    (h_norm_diff_tendsto :
      Tendsto (fun n : ℕ => ‖x (n+1) - x n‖) atTop (𝓝 0)) :
    ∃ (xstar : H), xstar ∈ (A + svEmbed B).zeros ∧
      Tendsto (fun k ↦ toWeakSpace ℝ H (x k)) atTop
        (𝓝 (toWeakSpace ℝ H xstar)) := by
  -- Proof omitted.
```

Figure 7: Revised statement for Theorem 2.1 of Moudafi & Oliny (2003). The two hypothesis names highlighted in green identify additions made during proof construction. The surrounding real Hilbert-space context and proof body are omitted.

D.2 Case 2: assuming properties that should be derived

The second example concerns Theorem 3.1 of Adly & Attouch (2023), represented by thm_2. For a convex potential and a solution of (DRYGRAD), the source establishes that the speed is nonincreasing and $\|\dot{z}(t)\| = o(1/t)$. Figure 8 compares the statement generated by PAPER2LEAN with the repaired statement. We judge the PAPER2LEAN output to be unfaithful because it treats intermediate properties as additional premises rather than deriving them from the source assumptions.

In particular, hvelocity_limit directly assumes $\dot{z}(t) \to 0$, and hA_nonneg supplies the nonnegative gradient-derivative pairing used in the velocity estimate. Both are consequences of the source assumptions in the proof: the former is obtained from the integrability result for the velocity, while the latter follows from convexity and the derivative of the gradient. The more substantial issue is the auxiliary vector field V. In the proof, V is constructed rather than passed as an input, namely

$$V(x) = -\,\mathrm{softThresholder}_r(\nabla f(x)).$$

The proof then derives its local Lipschitz property from the nonexpansiveness of soft thresholding and the local Lipschitz property of ∇f; proves hV_zero, the equivalence $V(x) = 0 \iff \|\nabla f(x)\| \le r$; and derives hvelocity from the DRYGRAD subdifferential inclusion. Finally, hstationary is proved using uniqueness for the ODE with this locally Lipschitz vector field, rather than assumed as an independent property. Thus V, its compatibility with the trajectory, its zero-set characterization, and the stationarity consequence are all intermediate objects or lemmas obtained inside the proof. Treating them as hypotheses changes proof obligations into extra restrictions on the theorem and can make the formalized result strictly weaker than the source theorem.

The repaired development uses gradient f and deriv z directly, derives the nonnegative pairing from convexity, and obtains velocity integrability from the preceding formalized Theorem 2.1. The preceding development also derives velocity convergence instead of requiring hvelocity_limit as an input. This repairs the dependency structure as well as simplifying

the interface. The repaired statement explicitly assumes C^2 regularity of f and z; the comparison demonstrates removal of the additional intermediate premises under that regularity, not unconditional equivalence of the two displayed statements.

(a) Automatic statement: additional intermediate premises

```
theorem theorem_3_1 {H : Type*} [NormedAddCommGroup H] [InnerProductSpace ℝ H]
    [CompleteSpace H]
    {t₀ r : ℝ} (hr : 0 < r) (f : H → ℝ) (fgrad : H → H) (z dz : ℝ → H)
    (hconvex : ConvexOn ℝ Set.univ f)
    (hgradient : ∀ x v, fderiv ℝ f x v = inner ℝ (fgrad x) v)
    (hsol : IsDrygradTrajectory r hr fgrad z dz t₀)
    (hf_lower : ∃ m : ℝ, ∀ x : H, m ≤ f x) (hfgrad_lip : LocallyLipschitz fgrad)
    (hinitial : r < ‖fgrad (z t₀)‖)
    (hchain : ∀ t ∈ Set.Ici t₀,
      HasDerivAt (fun s => f (z s)) (inner ℝ (fgrad (z t)) (dz t)) t)
    (hcomp_cont : ContinuousOn (fun t => f (z t)) (Set.Ici t₀))
    (V : H → H) (hV_lip : LocallyLipschitz V)
    (hV_zero : ∀ x, V x = 0 ↔ ‖fgrad x‖ ≤ r)
    (hvelocity : ∀ t ∈ Set.Ici t₀, dz t = V (z t))
    (hvelocity_limit : Filter.Tendsto dz Filter.atTop (nhds 0))
    (hstationary : ∀ S ∈ Set.Ici t₀, V (z S) = 0 →
      ∀ t ∈ Set.Ici S, z t = z S)
    (ddz : ℝ → H) (A : ℝ → H → H)
    (hdz_deriv : ∀ t ∈ Set.Ici t₀, HasDerivAt dz (ddz t) t)
    (hgrad_deriv : ∀ t ∈ Set.Ici t₀,
      HasDerivAt (fun s => fgrad (z s)) (A t (dz t)) t)
    (hA_nonneg : ∀ t ∈ Set.Ici t₀, 0 ≤ inner ℝ (A t (dz t)) (dz t)) :
    AntitoneOn (fun t => ‖dz t‖) (Set.Ici t₀) ∧
      Filter.Tendsto (fun t : ℝ => t * ‖dz t‖) Filter.atTop (nhds 0) := by
  -- Proof omitted.
```

(b) Human-repaired statement

```
theorem theorem_3_1 {H : Type*} [NormedAddCommGroup H] [InnerProductSpace ℝ H]
    [CompleteSpace H]
    {t₀ r : ℝ} (hr : 0 < r) (f : H → ℝ) (z : ℝ → H)
    (hf : ContDiff ℝ 2 f)
    (hconvex : ConvexOn ℝ Set.univ f)
    (hsol : IsDrygradTrajectory r f z t₀)
    (hf_lower : ∃ m : ℝ, ∀ x : H, m ≤ f x)
    (hgradient_lip : LocallyLipschitz (gradient f))
    (hz_contdiff : ContDiff ℝ 2 z)
    (hinitial : r < ‖gradient f (z t₀)‖) :
    AntitoneOn (fun t => ‖deriv z t‖) (Set.Ici t₀) ∧
      Filter.Tendsto (fun t : ℝ => t * ‖deriv z t‖) Filter.atTop (nhds 0) := by
  -- Proof omitted.
```

Figure 8: Statement repair for Theorem 3.1 of Adly & Attouch (2023). The highlighted names identify selected intermediate premises and the convexity assumption in the automatic version. Each panel uses its corresponding version of `IsDrygradTrajectory`; proof bodies are omitted.

E LIMITATIONS

First, the quality and efficiency of downstream formalization depend strongly on the extracted natural-language items, so the extraction stage remains a limitation. Missing assumptions scattered throughout a paper can cause proof construction to stall and waste tokens. Second, we have not compared our approach with other pipelines or studied how different harness architectures affect performance. Future work will evaluate a broader range of pipelines and harness designs. For example, `DeepSeek-V4.1-Flash` may perform better with the DeepSeek Harness (DeepSeek-AI, 2026).