Three Roads to de Finetti's Theorem in Lean 4

Cameron Freer
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We present a Lean 4 formalization of the de Finetti–Ryll-Nardzewski theorem for infinite sequences of random variables on standard Borel spaces, establishing that every exchangeable sequence is conditionally i.i.d. The development closely follows Kallenberg's modern treatment of probabilistic symmetries and formalizes three distinct proofs of the key implication, with the second and third formalized for real-valued square-integrable sequences: (i) a reverse-martingale argument due to Aldous, (ii) an elementary L^2 approach based on contractability bounds and Cesàro convergence, and (iii) an ergodic-theoretic proof via the Koopman operator and the mean ergodic theorem. The library contains over 42,000 lines of code and was completed in three months with extensive use of Claude and GPT models, together with a reusable Lean proof-engineering skill for agentic coding systems developed during the project. The three proofs share a uniform common ending, so each route had to produce the same finite conditional-factorization interface before the final conclusion. This provided a cross-check on these independent routes during AI-assisted development.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases exchangeability, de Finetti's theorem, Lean 4, formalized mathematics, AI-assisted formalization

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.34

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/cameronfreer/exchangeability` [7]
Software (Source Code): `https://github.com/cameronfreer/lean4-skills` [6]

Acknowledgements The author thanks Daniel M. Roy for suggesting this project and for advice throughout; and Mauricio Barba, Fabian Zaiser, Katherine Collins, McCoy Becker, Jason Rute, and Rémy Degenne for helpful conversations or feedback on drafts. Thanks also to users of the skill who have contributed bug fixes or feature implementations, and to the anonymous reviewers for comments that improved the paper and the formalization.

1 Introduction

Exchangeability is a key symmetry condition in probability: an infinite random sequence is exchangeable if its distribution is invariant under permutations of the index set. The classic de Finetti representation theorem states that such sequences are precisely those that are conditionally i.i.d. with respect to a directing random probability measure. We formalize a modern measure-theoretic form of this principle on standard Borel spaces [10, 14]. The theorem is a useful formalization target because it stresses several delicate interfaces at once: infinite random sequences, conditional laws as kernels, conditional independence, product-measure uniqueness, and L^1/L^2 convergence. Kallenberg's three short proofs make it a useful AI-assisted formalization benchmark: they share a final interface while requiring different bridges between symmetry, convergence, and conditional factorization.

This paper also serves as a case study in AI-assisted formalization of a large measure-theoretic development. Three proofs were formalized, closely following Kallenberg's text [10]. Unlike blueprint-driven formalizations, this project began from those short proofs rather than a detailed formal plan; auxiliary lemmas and interfaces were identified during the formalization. We used large language models (LLMs) and associated tooling as an interactive

copilot for such lemma discovery and other proof-structure suggestions, as well as for the proofs themselves ("filling sorries" once the scaffolding was in place). The resulting code was then reviewed, refactored, and simplified ("golfed"). The author directed the architecture and chose which generated edits to keep; accepted Lean code was generated by LLMs and mechanically checked by Lean.

We did not initially anticipate that the project would grow to its eventual size, since the source text containing all three proofs is only about 2.5 pages long. The size is due less to the theorem statement itself than to the bridge work needed to connect Kallenberg's informal arguments to existing mathlib APIs: reverse filtrations and tail σ-algebras, conditional-expectation bookkeeping, finite conditional-factorization statements, product-measure uniqueness, and transport between L^1, L^2, and kernel-based formulations. In practice the three routes required comparable effort but for different reasons: reverse-martingale and tail-σ-algebra infrastructure; elementary but complicated L^2 contractability bounds; and bridges from the mean ergodic theorem to the conditional-expectation identities required by the common ending. As in Kallenberg, all three proofs relied on a "common ending"; although this was fairly involved, we formalized it near the beginning of the project, and it was able to enforce a unified interface to what might have otherwise been even more disparate proofs.

Alongside the formalization, we recorded recurring patterns in a reusable Lean agent skill (used primarily through Claude Code in this project, and also exercised through Codex). Initially, this consisted of a reference library (mostly specific to measure theory) that was loaded incrementally on demand when editing Lean files. These reference files recorded recurring measure-theoretic proof patterns and failure modes in Lean, which in turn shaped repository conventions. Eventually, the skill grew to include guides to using the Lean LSP Model Context Protocol (MCP) server [5] (which was especially helpful for finding relevant results in mathlib), as well as proof workflows that encoded orchestration we initially performed by hand.

Contributions.

- We mechanize the de Finetti–Ryll-Nardzewski theorem in Lean 4 and expose it via a compact public API.
- We provide three mechanically checked proofs of the core implication, sharing a uniform common ending. The martingale proof covers general standard Borel spaces; the L^2 and Koopman proofs are formalized for real-valued, square-integrable sequences.
- We contribute a case study of human-LLM collaboration, including a reusable Lean agent skill (in a Claude Code-compatible format, also exercised through Codex) designed to support AI-assisted work on other Lean projects.

Artifact.

The first accompanying artifact [7] is a Lean 4.27.0-rc1 project (42,745 lines across over 100 files) that builds with a single `lake build` invocation against a pinned mathlib toolchain. Running `#print axioms deFinetti` confirms that no axioms beyond the standard three (`propext`, `Quot.sound`, `Classical.choice`) are used.

The second accompanying artifact [6] is the Lean agent skill that was developed in tandem with the project, and which has since been used in diverse Lean developments by the author and others.

Related formalizations.

de Finetti's original result for $\{0,1\}$-valued sequences [2] and later result for real-valued random variables [3] were generalized by Hewitt and Savage [8] to compact Hausdorff spaces, and by Ryll-Nardzewski [14], who established the further equivalence of contractability with exchangeability. Kallenberg's text [10] provides the treatment via three proofs that we follow.

To our knowledge, no prior Lean formalization of de Finetti's theorem exists. Perhaps the most closely-related project is the formalization by Ying and Degenne [20] of Doob's (forward) martingale convergence theorems in mathlib; our development provides reverse-martingale analogues, including Lévy's downward theorem, a time-reversed filtration construction, and a proof that the reversed process is a forward martingale. Another related project is the formalization by Degenne et al. [4] of Brownian motion in Lean. Some of its measure-theoretic infrastructure could likely be reused in the de Finetti project or conversely, and it is plausible that coordinated upstreaming to mathlib of conditional-expectation helpers, Kolmogorov extension machinery, or L^p convergence results could benefit both libraries.

LLM-assisted theorem proving.

Recent LLM-assisted theorem-proving systems play several roles different from ours. LeanDojo / ReProver [19] provides a Lean environment, benchmark, and retrieval-augmented prover; Lean Copilot [15] and LLMSTEP [17] integrate LLM proof-step suggestions into Lean; and AlphaProof [9] and APOLLO [13] use more specialized search or repair pipelines. Closer to our setting, COPRA [16] uses a general-purpose LLM with proof-state feedback for Lean and Rocq, CoqPilot [11] generates and checks Rocq proof candidates, and Agentic Proof Automation [18] and Numina-Lean-Agent [12] apply off-the-shelf coding agents to substantial Lean developments; Numina-Lean-Agent's setup guide also installs `lean4-skills`, an example of reuse beyond this project. Our contribution is complementary: we give a mechanically checked case study of human-steered agentic proof engineering in measure-theoretic probability, with three proof routes, a shared formal interface, and a reusable Lean agent skill developed during the project, rather than a new prover or benchmark.

2 Statement and formal interface

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and consider a sequence $X : \mathbb{N} \to \Omega \to \alpha$ of α-valued random variables. We assume both Ω and α are standard Borel spaces so that conditional laws can be represented by probability kernels (in particular, the assumption on Ω is a requirement for conditional-expectation infrastructure in Lean). We say that X is *exchangeable* if its distribution is invariant under arbitrary permutations of its indices; this turns out to be equivalent to invariance of finite-dimensional marginals under permutations that move finitely many indices (a statement which we also formalize). An ostensibly weaker condition is that of *contractability*, namely that all strictly increasing subsequences of equal length have the same distribution.

The main theorem of the library (see Figure 1) states that any contractable sequence (and hence in particular any exchangeable sequence) is conditionally i.i.d. with respect to a probability kernel representing the directing random measure. The library exposes three route-specific variants (e.g., `conditionallyIID_of_contractable_viaL2`), enabling users to choose the proof route and supporting infrastructure they depend on [7]. We also formalize the substantially easier converse, that any conditionally i.i.d. sequence is contractable.

```
theorem deFinetti_RyllNardzewski_equivalence
    [StandardBorelSpace Ω]
    {α : Type*} [MeasurableSpace α] [StandardBorelSpace α] [Nonempty α]
    {μ : Measure Ω} [IsProbabilityMeasure μ]
    (X : ℕ → Ω → α) (hX_meas : ∀ i, Measurable (X i)) :
    Contractable μ X ↔ Exchangeable μ X ∧ ConditionallyIID μ X
```

Figure 1 The full equivalence form of the de Finetti–Ryll-Nardzewski theorem, as stated in Lean.

3 One pipeline, three roads

Figure 2 gives a simplified contribution map; the full module-import and blueprint dependency graphs are included with the artifact. Table 1 gives codebase statistics by component.

3.1 Shared pipeline and the common ending

All three proof routes follow the same global pipeline: (symmetry) ⇒ (canonical σ-algebra) ⇒ (conditional factorization of finite blocks) ⇒ (directing probability kernel) ⇒ (conditionally i.i.d.). The martingale route uses tail σ-algebras and reverse martingale convergence; the L^2 route uses analytic convergence under square-integrability assumptions; and the mean-ergodic route uses invariance under a shift-induced Koopman operator [10].

Common ending. The *common ending* consumes a normalized intermediate statement: given a sub-σ-algebra $\mathcal{G}$ and a finite-dimensional conditional factorization hypothesis for $(X_0, \ldots, X_{n-1})$, it constructs a probability kernel K (with the required measurability relative to $\mathcal{G}$) and packages the conclusion as `ConditionallyIID`. Isolating this step also avoids repeating measurability and product-measure uniqueness arguments across all three routes. The common ending also plays a methodological role: three independent routes discharging the same finite conditional-factorization interface cross-check the routes, helping expose route-specific assumptions or definitions tailored to one proof during AI-assisted development.

3.2 Route A: reverse martingales (Aldous)

This is the only route that proves the core implication for general standard Borel target spaces, following Kallenberg's reverse-martingale argument due to Aldous [1]. Contractability supplies the symmetry relating conditional expectations along a reverse filtration, and reverse martingale convergence yields the desired conditional factorization. Since martingale convergence is stated for real-valued conditional expectations while the final statement concerns arbitrary measurable events and finite blocks of an α-valued sequence, the Lean proof reduces set-level statements to indicator functions and transports them back through measurable embeddings. The local infrastructure is therefore the bookkeeping needed to put reverse filtrations, tail σ-algebras, indicators, and standard-Borel transport into the interface consumed by the common ending.

3.3 Route B: L^2 proof (elementary bounds + Cesàro convergence)

Route B proves the same intermediate factorization under stronger hypotheses: real-valued, square-integrable sequences. The proof uses contractability bounds to control correlations and then applies L^2 convergence (often Cesàro-averaged) to obtain conditional factorization. Although Route B contains substantial local CDF/Stieltjes code (over 12,000 lines beyond

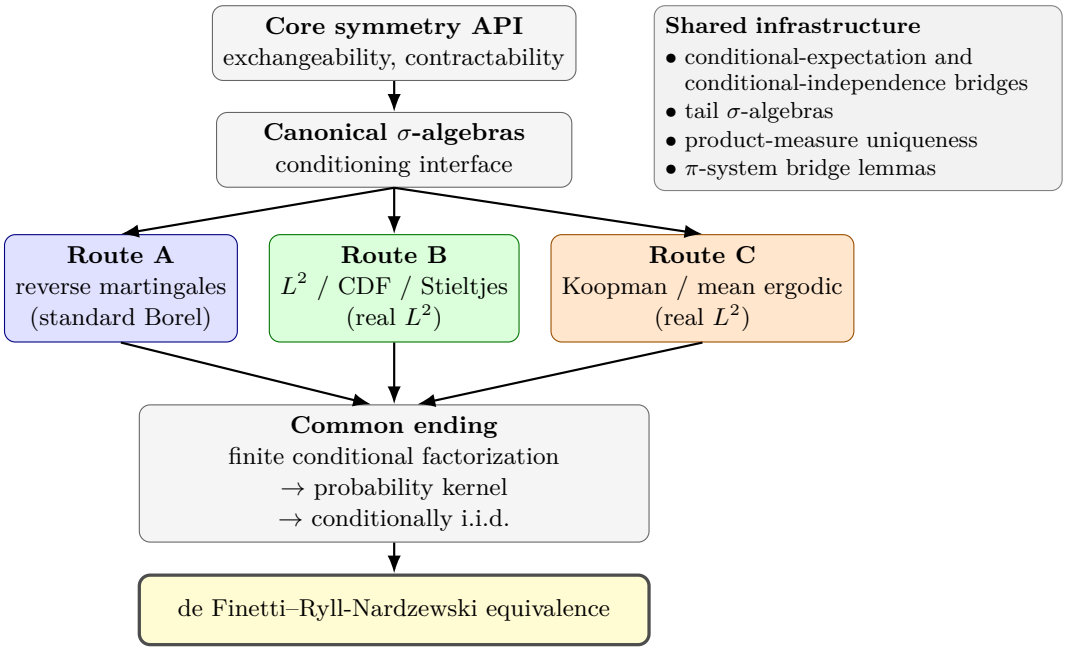

Figure 2 Simplified contribution map of the formalization. Arrows show proof flow. The shared-infrastructure box lists supporting module groups used across the routes and common ending; individual dependencies are omitted to reduce clutter. The full module-import and blueprint graphs are included with the artifact. Entries in the shared-infrastructure box name local APIs that wrap or extend existing mathlib infrastructure; for example, the π-system entry denotes local product-measure and finite-factorization lemmas built on mathlib's existing π-system API.

the shared machinery), this should not be read as a claim that mathlib lacked such notions. Mathlib already provides `StieltjesFunction`, real CDFs, measurable rational-CDF constructions, conditional CDFs, and CDF-to-kernel APIs. The local work was instead a bridge from the particular L^1 limits of empirical indicator averages produced by this proof to the common-ending interface: showing that the limiting rational-cut functions satisfy the required monotonicity, normalization, right-continuity, measurability in the conditioning variable, and finite-block factorization properties in the project's sub-σ-algebra/conditional-expectation setting. This cluster is therefore the main candidate for refactoring against mathlib's existing `MeasurableStieltjes`, `CondCDF`, and `CDFToKernel` APIs; after such a refactor, only the remaining generic bridge lemmas should be considered for upstreaming.

3.4 Route C: Koopman operator + mean ergodic theorem

Route C views the sequence on path space, where the shift induces a Koopman operator on an L^2 space; the mean ergodic theorem identifies the limit of Cesàro averages with the orthogonal projection onto the invariant subspace. The common ending, however, consumes L^1-style conditional-expectation identities for finite products of indicator functions rather than an operator-theoretic projection statement. The formal work in Lean is therefore concentrated in connecting the invariant subspace with the appropriate invariant σ-algebra, transporting L^2 convergence to set-integral identities for conditional expectation, and packaging them in the normalized form produced by the other two routes.

Table 1 Codebase statistics: lines of Lean, source files, and declarations (theorem/lemma/def/instance) [7]. Rows group local files by project role rather than novelty; the "Probability bridges" row consists mainly of wrappers, transport lemmas, finite-factorization statements, and adapters around existing mathlib probability APIs.

Component	Lines	Files	Decls
Route-specific			
Route A (`ViaMartingale`)	3,770	13	112
Route B (`ViaL2`)	12,476	12	96
Route C (`ViaKoopman`)	6,893	18	138
Shared infrastructure			
Common ending & bridge	917	2	29
Theorem files (API)	603	4	15
Probability bridges (cond. exp., cond. indep.)	10,133	36	156
Ergodic theory	1,959	6	59
Tail σ-algebras	1,214	3	21
Core (exchangeability, contractability)	1,536	3	68
Other (helpers, utilities, path space)	4,011	15	89
Total	**42,745**	**112**	**783**

Assessment of the generated library.

The artifact should be viewed as a checked research formalization rather than mathlib-ready code. Its public API is small: the main equivalence theorem, route-specific variants, and the definitions `Exchangeable`, `Contractable`, and `ConditionallyIID`; most of Table 1 is proof infrastructure. We classify the local development into four categories: direct mathlib replacement candidates; wrappers or adapters around existing APIs; reusable bridge lemmas likely suitable for staged upstreaming after refactoring; and exploratory route-specific code, especially local CDF/Stieltjes and kernel-construction code in Route B. The probability-related files are not a replacement for mathlib's conditional-probability infrastructure: they are mainly wrappers, transport lemmas, finite-factorization statements, and adapters around existing mathlib definitions, and some may duplicate mathlib results under different formulations. Accordingly, the first refactoring pass should not be upstreaming but triage: replace or delete statements already covered by mathlib, keep only necessary adapters, and factor out genuinely reusable bridge lemmas. The development builds reproducibly on the pinned toolchain and uses only standard axioms, but candidate files will need naming cleanup, import minimization, and performance checks on the slowest generated proofs before upstreaming.

4 A reusable Lean agent skill developed during the project

During the project, we developed a reusable *Lean agent skill*: a package of proof-engineering guidance, examples, reference files, command recipes, prompts, and helper scripts for agentic coding systems working in Lean. The artifact packages this material in a Claude Code-compatible `SKILL.md` layout, because that was the format used most heavily during the formalization. The content, however, is not Claude-specific: most of it consists of plain-text Lean/mathlib workflows, shell commands, proof-repair patterns, review checklists, and repository-local scripts. We also exercised the same content in Codex-style agentic workflows.

The skill [6] was developed concurrently with the formalization: recurring proof patterns and failure modes were recorded as guidance for later sessions. It now contains 36 domain

references, 6 command recipes (invokable as slash commands such as `/prove` or `/golf`), 4 subagent prompts (role-specific instruction templates for separate coding-agent invocations, such as proof repair, review, search, or refactoring), and associated hooks and scripts.

The largest references target recurring failure modes seen with generic coding agents. The measure-theory reference codifies sub-σ-algebra instance management, conditional-expectation idioms, and the "instance pollution" failure mode that caused repeated timeouts. The domain-patterns reference catalogues proof patterns across measure-theoretic tasks. The compilation-errors reference maps common Lean error messages to targeted fixes.

A second component of the skill documents use of the Lean LSP MCP server [5]; the skill exposes Lean LSP diagnostics to the coding agent, playing a role analogous to Lean's VS Code InfoView, the editor panel used to inspect goals, hypotheses, diagnostics, and related widgets. The skill also guided the agent toward mathlib search endpoints, which was important for avoiding local reimplementation.

5 Conclusion

We presented an AI-assisted Lean 4 formalization of the de Finetti–Ryll-Nardzewski theorem with three mechanically checked proof routes sharing a common finite conditional-factorization interface. This shared interface helped expose drift in definitions or supporting lemmas before it propagated to the final theorem statement. The artifact is a research formalization, not mathlib-ready code: its public API is small, and most of the 42k lines are bridge lemmas, route-specific infrastructure, and adapters around mathlib probability APIs. Upstreaming will be staged: after refactoring against mathlib, conditional-expectation and σ-algebra bookkeeping could go first, then tail/reverse-martingale infrastructure, then product-measure/factorization bridges; larger APIs such as the conditionally-i.i.d. interface would wait until their formulation aligns with mathlib conventions. We also developed a reusable Lean agent skill (packaged for Claude Code, also exercised through Codex), which has already begun to aid other AI-assisted Lean formalization projects.

GenAI disclosure

In preparing this paper, Claude Opus 4.5/4.6 was used to collate notes from approximately 4,000 commit messages in the project repository in order to track the proof-development history. GPT Codex 5.3, GPT 5.5, and Claude 4.7 were used to review the text and propose improvements.

The formalization artifact [7] was developed through a human-steered LLM-assisted workflow. Candidate Lean edits were generated primarily by Claude models (Sonnet 4/4.5 and Opus 4.5) in Claude Code, using command-line access to Lean and the Lean LSP MCP server [5], guided by the Lean agent skill described in Section 4. The author directed the proof architecture, selected the mathematical interfaces, supervised the main definitions and theorem statements, and decided which generated changes to keep. LLM assistance, primarily GPT Codex with occasional mathematical assistance from GPT Pro, was also used to review and repair generated code. The accepted proof scripts in the artifact are mechanically checked by Lean on the pinned toolchain.

The Lean skill artifact [6] was written under the author's direction using Sonnet 4/4.5 and Opus 4.5 in Claude Code to collate notes, revise content, and describe workflow automation.

References

1 David J. Aldous. Exchangeability and related topics. In *Ecole d'Ete de Probabilites de Saint-Flour XIII — 1983*, volume 1117 of *Lecture Notes in Mathematics*, pages 1–198. Springer, 1985. doi:10.1007/BFb0099421.

2 Bruno de Finetti. Funzione caratteristica di un fenomeno aleatorio. *Atti della Reale Accademia Nazionale dei Lincei, Serie VI, Memorie*, 4(5):86–133, 1930. URL: http://www.brunodefinetti.it/Opere/funzioneCaratteristica.pdf.

3 Bruno de Finetti. La prévision: ses lois logiques, ses sources subjectives. *Annales de l'Institut Henri Poincaré*, 7(1):1–68, 1937. URL: https://eudml.org/doc/79004.

4 Rémy Degenne, David Ledvinka, Etienne Marion, and Peter Pfaffelhuber. Formalization of Brownian motion in Lean. arXiv preprint, 2025. URL: https://arxiv.org/abs/2511.20118.

5 Oliver Dressler. Lean Theorem Prover MCP. GitHub repository, 2025. URL: https://github.com/oOo0oOo/lean-lsp-mcp.

6 Cameron Freer. lean4-skills: Lean 4 coding-assistant skill for Claude Code. GitHub repository, 2025. URL: https://github.com/cameronfreer/lean4-skills.

7 Cameron Freer. exchangeability: Exchangeability and de Finetti's theorem in Lean 4, v1.2. GitHub repository, 2026. URL: https://github.com/cameronfreer/exchangeability.

8 Edwin Hewitt and Leonard J. Savage. Symmetric measures on Cartesian products. *Transactions of the American Mathematical Society*, 80(2):470–501, 1955. doi:10.1090/S0002-9947-1955-0076206-8.

9 Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi:10.1038/s41586-025-09833-y.

10 Olav Kallenberg. *Probabilistic Symmetries and Invariance Principles*. Probability and Its Applications. Springer, 2005. doi:10.1007/0-387-28861-9.

11 Andrei Kozyrev, Gleb Solovev, Nikita Khramov, and Anton Podkopaev. CoqPilot, a plugin for LLM-based generation of proofs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024)*, 2024. doi:10.1145/3691620.3695357.

12 Junqi Liu, Zihao Zhou, Zekai Zhu, Marco Dos Santos, Weikun He, Jiawei Liu, Ran Wang, Yunzhou Xie, Junqiao Zhao, Qiufeng Wang, Lihong Zhi, Jia Li, and Wenda Li. Numina-Lean-Agent: An open and general agentic reasoning system for formal mathematics. arXiv preprint, 2026. URL: https://arxiv.org/abs/2601.14027.

13 Azim Ospanov, Farzan Farnia, and Roozbeh Mohit. APOLLO: Automated LLM and Lean collaboration for advanced formal reasoning. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. URL: https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b77109ad4dd4ba82d07cacd4b24207e-Abstract-Conference.html.

14 Czesław Ryll-Nardzewski. On stationary sequences of random variables and the de Finetti's equivalence. *Colloquium Mathematicum*, 4(2):149–156, 1957. doi:10.4064/cm-4-2-149-156.

15 Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean Copilot: Large language models as copilots for theorem proving in lean. In George Pappas, Pradeep Ravikumar, and Sanjit A. Seshia, editors, *Proceedings of the International Conference on Neuro-symbolic Systems*, volume 288 of *Proceedings of Machine Learning Research*, pages 144–169. PMLR, 2025. URL: https://proceedings.mlr.press/v288/song25a.html.

16 Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. In *Conference on Language Modeling (COLM)*, 2024. URL: https://openreview.net/forum?id=V7HRrxXUhN.

17 Sean Welleck and Rahul Saha. LLMSTEP: LLM proofstep suggestions in Lean. arXiv preprint, 2023. URL: https://arxiv.org/abs/2310.18457.

18 Yichen Xu and Martin Odersky. Agentic proof automation: A case study. arXiv preprint, 2026. URL: https://arxiv.org/abs/2601.03768.

19 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. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

20 Kexing Ying and Rémy Degenne. A formalization of Doob's martingale convergence theorems in mathlib. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023)*, 2023. doi:10.1145/3573105.3575675.