Live Lean Triathlon: A Live-updating Three-Mode Benchmark in Formal Theorem Proving

Yichuan Cao[*], Ruichen Qiu[*], Bolton Bailey[*], Junqi Liu[*], Yunzhou Xie[*], Justin Asher[*], Zekai Zhu[*], Felix Pernegger[*], Kalle Kytölä[†], Jiayang Hong[*], Krsto Proroković[*], Weikun He[*], Jiatong Yang[*], Yizheng Zhu[*], Zihao Zhou[*], Jia Li[*], Wenda Li

Abstract

The integration of Large Language Models (LLMs) with formal verification systems like Lean has opened new frontiers for automated mathematical reasoning. However, existing benchmarks often treat theorem proving as static, isolated tasks and suffer from rapid performance saturation, failing to capture the varying degrees of mathematical guidance present in realistic formalization workflows. In this paper, we introduce *Live Lean Triathlon*, a novel Lean benchmark consisting of 110 formalized statements spanning undergraduate to research-level mathematics. Mirroring the grueling, multi-stage nature of a triathlon, our framework comprehensively evaluates AI agents across three distinct modes of guidance: (1) *Bare-Statement*, which demands entirely autonomous, full proof discovery; (2) *Sketch-Guided*, which provides a structured framework of formally stated intermediate lemmas; and (3) *Blueprint-Guided*, which challenges models to accurately translate high-level natural language intuition into executable Lean code. To ensure long-term utility, the benchmark is designed to be live-updating, with a scheduled expansion every six months that introduces new formalized statements and their accompanying informal proofs. Furthermore, we introduce a folder-level verification protocol that evaluates interdependent Lean files within the full project context layered on Mathlib, moving beyond isolated, context-free snippets. *Live Lean Triathlon* establishes a rigorous, multi-dimensional proving ground to assess cross-domain generalization and fine-grained capabilities of next-generation AI mathematical agents.

1 Introduction

The idea of using computers to emulate human mathematical reasoning has long intrigued the computer science [6]. The Lean theorem prover [12] offers a concrete platform for this vision, allowing mathematical propositions and proofs to be expressed as executable code. With the rise of Large Language Models (LLMs) [14] and coding agents, there has been growing interest in using AI to assist formalization and, conversely, using formal theorem proving as a benchmark for AI capabilities [25]. The Lean community has particularly benefited from Mathlib [19], a library covering undergraduate and graduate mathematics, with tools that connect AI agents to the Lean environment [4, 10].

Formal proof generation is an ideal evaluation task because outputs are easily verifiable by a proof assistant (Lean provides the 'comparator' system [18] for verification). However, in realistic formalization workflows, an AI model may receive varying degrees of mathematical guidance. For example, it could be given only the formal statement of a theorem, or a set of formally stated intermediate lemmas, or an informal proof (a "blueprint") to be translated into Lean. These scenarios demand different skills: proof search, decomposition,

[*]Project Numina
[†]Aalto University

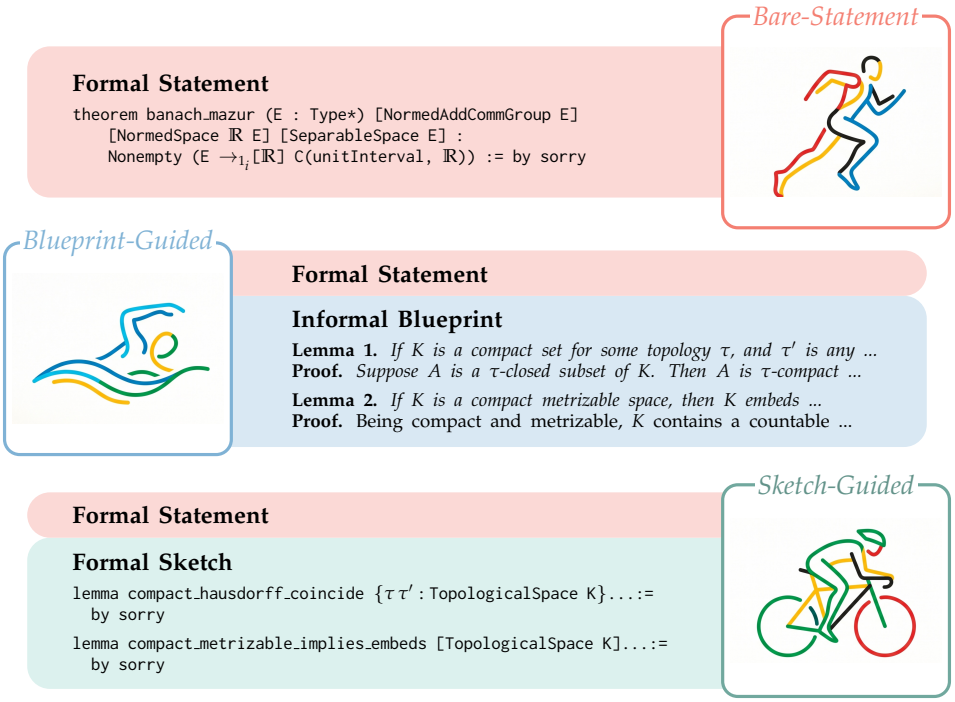


Figure 1: Live Lean Triathlon. Inspired by the multi-stage nature of a triathlon, we design three evaluation modes to comprehensively benchmark theorem proving capabilities.

library retrieval, and autoformalization. Moreover, as LLM-based provers improve, static benchmarks quickly suffer from saturation – the best models solve all problems, making differentiation difficult. Leakage into training corpora further undermines their reliability.

To address these issues, we introduce a new Lean dataset and benchmark named *Live Lean Triathlon*, consisting of 110 formalized statements drawn from diverse mathematical fields. Inspired by the grueling and multi-stage nature of a triathlon, we design three distinct evaluation modes to comprehensively benchmark a model's capabilities, as illustrated in Figure 1: **(1)** ***Bare Statement***: Only the formal statement is provided, requiring the model to achieve entirely autonomous full proof discovery. **(2)** ***Sketch-Guided***: The model receives the formal statement along with a structured framework of formally stated intermediate lemmas (a formal sketch) to guide the proving process. **(3)** ***Blueprint-Guided***: The model is equipped with the formal statement paired with an informal proof, challenging it to accurately translate high-level mathematical intuition into executable Lean code.

By providing tasks of varying difficulty levels across diverse disciplines, the *Live Lean Triathlon* benchmark is inherently resilient against performance saturation. As automated theorem proving models continue to evolve, evaluation can dynamically pivot toward harder, unproven objectives to yield fine-grained distinctions in agent capabilities, while the inclusion of multiple mathematical fields enables a rigorous assessment of cross-domain generalization. Our main contributions are as follows:

A Comprehensive and live-updating Lean benchmark. We introduce a new mathematical dataset spanning undergraduate to research-level topics. A core subset of these problems features complete formal proofs paired, adopting a background-lemma architecture where key intermediate steps are formally represented. The benchmark is designed to be live-updating, with a scheduled expansion every six months that introduces new formalized statements and their accompanying informal proofs.

The tri-Mode evaluation suite. We establish the *Live Lean Triathlon* evaluation framework to systematically measure model performance under three distinct levels of guidance: *Bare-Statement*, *Sketch-Guided* and *Blueprint-Guided*.

Table 1: Comparison of representative formal mathematics datasets and benchmarks. *Live Lean Triathlon* strikes an optimal balance across key architectural dimensions, distinguishing itself through a broad multi-domain mathematical scope, a tri-mode evaluation suite, realistic folder-level verification, and a continuous living update protocol. Notation: P for proving, BGP for blueprint-guided proving, SGP for sketch-guided proving.

Dataset	Size	Scope	Verification	Eval Mode	Updating
miniF2F	488	Level Specific (High-school)	Snippet-level	P	✗
ProofNet	371	Level Specific (Undergrad)	Snippet-level	P+BGP	✗
PutnamBench	640	Level Specific (Undergrad)	Snippet-level	P	✗
FormalMath	5,560	Level Specific (Undergrad)	Snippet-level	P	✗
ProofBench	200	Level Specific (Undergrad / Grad)	Snippet-level	P	✗
LeanCat	100	Domain specific (Category theory)	Snippet-level	P	✗
CombiBench	100	Domain specific (Combinatorics)	Snippet-level	P+BGP	✗
FATE	350	Domain specific (Algebra)	Snippet-level	P+BGP	✗
Lean-Eval	60	Broad scope	Snippet-level	P+BGP	✓
Live Lean Triathlon	110	Broad scope	Folder-level	P+BGP+SGP	✓

Folder-level verification protocol. We design a realistic evaluation protocol tailored for interdependent Lean files. Because our benchmark constructs its own cohesive library layered on top of Mathlib, all model submissions must be verified and compiled within the full project context rather than as isolated, context-free snippets.

2 Related Work

As Lean 4 has emerged as one of the most active and widely used theorem-proving ecosystems, several benchmarks have been developed to evaluate the formalization and proving capabilities of AI models. We systematically compare these existing datasets with *Live Lean Triathlon* across multiple structural dimensions in Table 1.

Evaluation Modes and Verification Protocols. An early pioneer in this space is miniF2F [25], which focuses primarily on high school math competitions and has undergone multiple revisions to correct errors [25, 13]. Closely related to our work, Jiang et al. [7] introduced the "Draft, Sketch, and Prove" paradigm, augmenting miniF2F with informal proofs. Their work successfully demonstrated the value of combining informal and formal reasoning. Building upon these foundations, several subsequent benchmarks have successfully expanded the evaluation framework to dual modes, combining statement-only proving (P) and blueprint-guided proving (BGP). For a more fine-grained diagnosis of agent capabilities, *Live Lean Triathlon* further advances this progression by establishing a comprehensive tri-mode spectrum that seamlessly integrates the previous two modes with the sketch-guided proving (SGP). Despite the evaluating modes, a fundamental architectural gap remains: all these existing benchmarks are strictly constrained to snippet-level verification of isolated theorems. To address this limitation, our benchmark pivots to a realistic folder-level verification protocol. By forcing model submissions to compile within a cohesive project context layered on top of Mathlib, *Live Lean Triathlon* faithfully captures real-world mathematical workflows featuring complex cross-file dependencies.

Mathematical Difficulty and Domain Breadth. Since the development of miniF2F, subsequent benchmarks have expanded to cover standard educational curricula [3, 24, 15], with recent efforts like NuminaMath [8, 22] scaling dataset sizes up to 100K data points. Despite their impressive volume, these problems remain predominantly restricted to level specific topics, making them highly susceptible to rapid performance saturation. In contrast, domain specific benchmarks have emerged to capture higher-level complexity, such as LeanCat [23] for category theory and CombiBench [9] for combinatorics. While effective in capturing the depth of real-world formalization projects, their narrow focus limits their ability to evaluate how well an AI agent generalizes across disparate mathematical fields. *Live Lean Triathlon* strikes an ideal balance by prioritizing breadth over depth across advanced topics, spanning undergraduate to research-level mathematics while retaining cross-domain diversity.

Dynamic Evolution and Living Updates. A critical yet rarely addressed vulnerability of existing theorem-proving datasets is their static nature, which exposes them to severe training data contamination and rapid performance saturation over time. To the best of our knowledge, within the Lean ecosystem, only *Lean-eval* [1] introduces a living update mechanism to counteract these issues. Such dynamic updates are essential not only for mitigating data leakage as LLMs iterate, but also for maintaining alignment with Mathlib's continuous evolution. Therefore, one core desideratum of our data curation is evolvability.

3 Data Curation

In this section, we present *Live Lean Triathlon*, a benchmark designed to evaluate LLM agents on formal mathematical proving. We first outline the core desiderata and constraints that guide our dataset design. We then detail our data sources and the interactive collection methodology used to formalize these problems. Next, we provide summary statistics and analyze the structural distribution of the dataset based on the AMS Mathematics Subject Classification. Finally, we describe our protocol for the ongoing evolution and maintenance, ensuring it remains dynamic and resistant to performance saturation over time.

3.1 Design Principles and Desiderata

In sourcing data for the benchmark, we navigate several key desiderata. *(1) Data Sourcing.* To avoid data contamination, we prioritize problems without existing formal proofs online, as LLMs may exploit leaked solutions or retrieve them dynamically. We extensively search for prior formalizations and explicitly flag any subsequently formalized by external efforts. Public release is restricted to prevent automated harvesting. *(2) Difficulty.* Trivial problems are excluded due to low discriminative power, while problems with prohibitive formalization overhead are avoided to ensure scalability. At minimum, each problem must be formally statable—a process that often reveals underlying proof complexity. *(3) Representativeness.* The problems must reflect real-world mathematical formalization challenges. Unlike olympiad-heavy benchmarks, our tasks span diverse disciplines and are selected to serve as lemmas or intermediate steps for broader applications.

3.2 Data Sources

To construct a comprehensive and diverse evaluation set, we curated a total of 110 theorems from multiple distinct mathematical ecosystems, shown in Figure 2 (b). Rather than relying on a single repository, we diversified our sources to ensure broad mathematical coverage and varying degrees of conceptual difficulty.

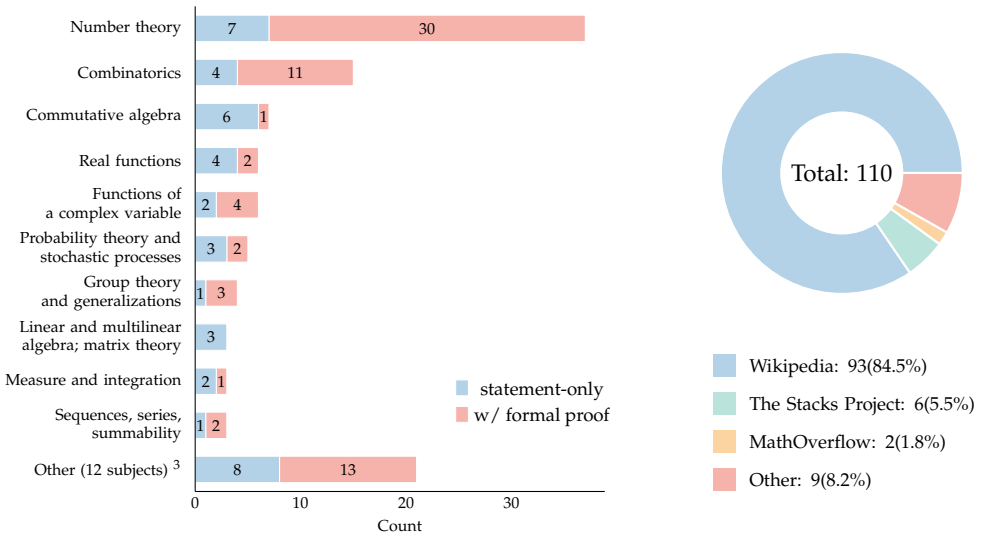
Wikipedia and the 1000+ Theorems Project (93 Theorems). The vast majority of our dataset is derived from Wikipedia and Wikidata, which aggregate historically and mathematically significant theorems. Topics documented must satisfy strict notability guidelines, guaranteeing that the included problems are of genuine interest to the mathematical community. To systematically prevent data contamination, we cross-referenced these candidates with the 1000+ Theorems Project [17], a community effort tracking formalized mathematics in Lean. Additionally, we expanded our search to include notable theorems listed on Wikipedia but not yet actively tracked by the 1000+ Theorems Project index, manually verifying their formalization status across public Lean repositories.

The Stacks Project (6 Theorems). The Stacks project [20] describes itself as an "open source textbook and reference work on algebraic geometry." The project is well-documented and rigorously written, making it relatively easy to assess the requirements for formalizing theorems.

MathOverflow (2 Theorems). To reflect the types of problems active researchers encounter, we selected high-quality entries from MathOverflow[2], a premier question-and-answer

[1] `https://lean-lang.org/eval/`

[2] `https://mathoverflow.net`



(a) Distribution across AMS Mathematics Subject Classification (b) Distribution across source

Figure 2: Distribution of theorems in LiveProveBench.

platform for professional mathematicians. Sourcing from this platform ensures that the selected problems capture nuances that mathematicians find sufficiently challenging or important to warrant collaborative research.

Community-Driven Sourcing (9 Theorems). Finally, a subset of problems was sourced from ongoing formalization discussions within the Lean community. These entries represent high-interest research vectors that are currently being pioneered by domain experts.

3.3 Collection and Formalization Pipeline

The dataset construction followed a structured, multi-stage pipeline designed to transition mathematical statements from informal descriptions to verified, agent-ready formal specifications. This pipeline can be partitioned into three primary phases: Identification, Formalization, and Iterative Refinement.

Phase 1: Candidate Identification and Filtering. We initiated the pipeline by leveraging the 1000+ Theorems Project database alongside public Lean community trackers to filter out already-formalized mathematics. For unformalized candidates, we utilized Large Language Models (LLMs) via the LeanExplore framework [1] to conduct preliminary dependency audits. This allowed us to determine whether the prerequisites for a given theorem were sufficiently supported by the current state of Lean's mathlib. Candidates passed this phase only if their foundational primitives were deemed feasible to formalize without requiring an extensive rewrite of core mathlib libraries.

Phase 2: Formal Specification and Expository Drafting. Once a candidate was validated, we manually constructed its formal statement in Lean, implementing any necessary definitions not currently present in mathlib. Concurrently, we authored a highly structured and self-contained informal proof. This expository text was designed to strike an optimal balance between mathematical rigor and concise logical progression, serving as the primary contextual prompt for subsequent automation.

Phase 3: Agentic Execution and Iterative Alignment. Using the formal statement and the drafted informal proof as input, we deployed a pre-existing LLM agent to attempt the for-

[3]General topology, Mathematical logic and foundations, Field theory and polynomials, Nonassociative rings and algebras, Functional analysis, Geometry, Convex and discrete geometry, Computer science, Algebraic geometry, Category theory; homological algebra, Operator theory, Calculus of variations and optimal control; optimization

malization. We then systematically audited the failure modes of the agent's attempts. If the agent failed due to syntactic or semantic errors in our initial statement, we corrected the formal definition. If the failure stemmed from logical gaps or ambiguities in the informal text, we refined the informal proof by decomposing complex arguments into explicit intermediate lemmas or purging orthogonal, non-essential steps. This alignment process was repeated iteratively until a stable benchmark entry was established.

Handling Structural Feasibility and Stubs. In practice, due to latent gaps in mathlib or unforeseen technical complexities in the proofs, certain theorems demanded a prohibitive amount of manual engineering to formalize fully. To maintain scalability without sacrificing the conceptual depth of the benchmark, we opted to include these entries as formalized stubs. For these instances, the theorem statements are fully formalized, but unfinished or highly intricate intermediate lemmas are explicitly marked with the Lean keyword `sorry`. This hybrid approach allows the benchmark to accommodate problems with accessible formulations but highly sophisticated proofs, directly shaping the structural composition of our dataset analyzed in the following section.

3.4 Dataset Statistics and Analysis

The *Live Lean Triathlon* benchmark comprises 110 formalized theorems. For 41 of these, we provide complete formal proofs alongside informal blueprints. The remaining 69 are "candidate" problems with one or more `sorry` placeholders in their proofs. These candidates are too challenging to resolve within current resources and serve as a reservoir of harder tasks for future evaluation. To evaluate the domain diversity of the benchmark, each theorem is tagged with its primary AMS Mathematics Subject Classification (MSC2020) [2], leveraging infrastructure from the formal-conjectures project [5]. While generally adopting the classifications from the 1000+ Theorems Project, we manually assigned missing tags.

Figure 2 illustrates the disciplinary distribution of the benchmark. The 110 theorems span 22 distinct high-level AMS classifications. Number theory (37 theorems) and combinatorics (15 theorems) are most heavily represented, yet exhibit low proof completion rates. This distribution occurs because the core primitives of number theory and combinatorics are comprehensively established within `mathlib`, rendering their theorem *statements* trivial to formalize; however, their underlying *proofs* remain remarkably difficult, leading to a high concentration of formalized stubs.

Beyond these heavily represented fields, certain AMS categories posed systemic obstacles. Theorems from game theory and behavioral sciences (AMS 91) in the 1000+ Theorems Project often lack sufficient precision in the literature, introducing semantic ambiguity for formalization. Applied physics fields, such as mechanics (70), quantum theory (81) and statistical mechanics (82), require rigorous formalization of foundational physical concepts before statements can be expressed. Although domain-specific frameworks such as *Lean-Quantuminfo* [11] and *Physlib* [21] exist, we deliberately avoided sourcing from these disciplines to minimize external dependencies.

3.5 Data Evolution and Maintenance

While the preceding sections detail the initial construction of *Live Lean Triathlon*, the benchmark is intentionally designed to be live-updating over time. We implement a sustainable evolution protocol to counter the static nature that plagues most existing theorem-proving datasets. Specifically, every six months, we will expand the benchmark by introducing 20 new formalized statements, each accompanied by its corresponding informal proof. For a designated subset of these problems, we will also provide complete formal proofs. This regular, rolling expansion serves two critical purposes: first, it mitigates the risk of training data contamination as LLMs evolve; second, it ensures that the benchmark remains aligned with the continuous evolution of Mathlib. By systematically refreshing the problem suite, *Live Lean Triathlon* guarantees that our evaluation remains a pristine, non-saturated, and highly challenging proving ground for next-generation AI agents, thereby avoiding the rapid performance saturation observed in static datasets.

4 Benchmarking Methodology

4.1 Tri-Mode Evaluation Suite

In order to comprehensively and multi-dimensionally evaluate the capability boundaries of different models, we design a triathlon-style, multi-mode evaluation framework. By considering various perspectives of mathematical assistance and prior guidance, we decouple each holistic proof task into three distinct, progressively challenging testing modes.

Sketch-Guided Mode. To evaluate the capabilities in local tactic synthesis and sub-goal solving, we design the Sketch-Guided mode, where the model receives a "formal proof skeleton" with intermediate lemmas, while all proof details are masked as `sorry`. These pre-stated lemmas act as a formal sketch, directly exposing the structural milestones within the proof process. With this predefined trajectory, the challenge for the model is to proficiently invoke low-level tactics to bridge the remaining logical gaps between these milestones. Furthermore, since the provided lemmas serve as a mutable working environment rather than black-box facts, this setting also assesses the model's flexibility to refine or reconstruct intermediate steps when necessary.

Blueprint-Guided Mode. Compared to the *Sketch-Guided* mode, this mode provides the model with a detailed, human-expert-verified natural language proof instead of a formal sketch. This text serves as a guiding blueprint, supplying the model with a complete mathematical problem-solving intuition. This mode primarily evaluates the model's capabilities in auto-formalization and cross-modal alignment. Specifically, it requires the agent not only to accurately comprehend high-level human mathematical intuition, conceptual abstractions, and proof logic, but also to precisely translate and ground them into rigorous, executable Lean formalization code.

Bare-Statement Mode. As the most demanding and rigorous evaluation setting, this mode removes all intermediate lemma hints and informal textual assistance. The model is presented solely with the isolated target theorem statement and the foundational concepts necessary to define it. This mode aims to probe the limits of the model's fully autonomous, end-to-end proof search capabilities. Without any external scaffolding or blueprint navigation, the model must demonstrate macroscopic global planning skills, including independently resolving missing dependencies, dynamically synthesizing its own proof skeletons, and iteratively refining its strategies based on compilation errors.

4.2 Folder-Level Verification Protocols

To accurately evaluate mathematical formalizations involving extensive cross-file dependencies, our evaluation for *Live Lean Triathlon* departs from traditional single-file server verification [16] in favor of a robust folder-level local verification mechanism. Since single-file evaluators often struggle to parse adaptive multi-file structure, we compile the target files directly within a complete local Lean 4 environment. To precisely monitor the compilation status, we developed a dedicated CLI tool that parses the raw compiler output, accurately extracting syntax/logic errors, unresolved `sorry` placeholders, and warnings, thereby providing high-fidelity environmental feedback for the agents and ensuring rigorous evaluation.

Furthermore, given the advanced autonomous capabilities of modern LLM agents, preventing unintended shortcuts or cheating is crucial to ensure the fairness of the benchmark. We instantiate isolated, independent workspaces for each testing task to prevent state contamination. More importantly, we enforce strict sandbox restrictions on the execution environment: we completely block access to network search, disable version control commands, and heavily restrict file system read/write permissions. These comprehensive isolation protocols physically prevent the models from web searching or fetching external repositories, forcing them to rely exclusively on their intrinsic reasoning abilities and the explicitly provided context.

Table 2: Evaluation statistics and normalized scores for the 20 selected theorems in the evaluation subset. The table is sorted in difficulty score and divided into three tiers. The values of code scale are color-coded to indicate different levels of complexity, while extreme cases of Mathlib coverage are marked with an underscore.

Name / Subject	L_{eff}	D_{f}	C_{m}	D_{m}	$D \uparrow$
DifferenceSetContainsInterval / Measure and integration	279	0.4	0.80	0.3	0.90
Kraft / Computer science	279	0.3	0.55	0.2	0.95
PicardGroupUfd / Commutative algebra	161	0.5	0.85	0.4	1.05
CondorcetJury / Probability theory and stochastic processes	649	0.5	0.35	0.2	1.35
Worpitzky / Combinatorics	1548	0.5	0.30	0.2	1.40
Dilworth / Combinatorics	629	0.5	0.35	0.3	1.45
JacobsonMorozovTheorem / Nonassociative rings and algebras	290	0.5	0.55	0.5	1.45
BanachMazur / Functional analysis	885	0.6	0.55	0.5	1.55
KolmogorovThreeSeriesNecessary / Sequences, series, summability	333	0.6	0.55	0.5	1.55
SchurRamsey / Combinatorics	31	0.5	0.20	0.3	1.60
MertensFirstTheorem / Number theory	15	0.5	0.20	0.3	1.60
SinkhornTheorem / Linear and multilinear algebra; matrix theory	1232	0.5	0.35	0.5	1.65
VonStaudtClausen / Number theory	907	0.6	0.35	0.4	1.65
AndersonTheorem / Real functions	1093	0.6	0.35	0.6	1.85
BrascampLieb / Real functions	1009	0.6	0.35	0.6	1.85
PolygonalNumberTheorem / Number theory	300	0.5	0.45	0.8	1.85
PicardTheorem / Functions of a complex variable	14	0.6	0.15	0.5	1.95
KaplanskyTheoremOnQuadraticForms / Number theory	18	0.6	0.10	0.6	2.10
DefinabilityNaturalsInRationals / Mathematical logic and foundations	19	0.6	0.15	0.8	2.25
FareySequence / Number theory	32	0.7	0.10	0.8	2.40

5 Experiments and Analysis

We present an evaluation of *Live Lean Triathlon* through three components. In Section 5.1, we describe the subset selection for the evaluation, where 20 representative problems are selected using a multi-dimensional pipeline based on effective code scale, Mathlib coverage, formalization difficulty, and mathematical depth, aggregated into an assembly difficulty score to ensure balanced coverage. We present the results of model performance in Section 5.2, and perform a brief analysis of three evaluation modes in Section 5.3.

5.1 Evaluation Subset Selection

Since the arge scale of the *Live Lean Triathlon* dataset (totaling 110 propositions), to ensure a comprehensive evaluation of model performance while effectively managing computational overhead, we construct a representative test subset comprising 20 problems. To guarantee that this subset is uniformly distributed across difficulty gradients and broadly covers diverse mathematical branches, we design a multi-dimensional automated evaluation pipeline to conduct unbiased quantitative profiling of all 110 candidate problems across the following three core dimensions.

Core Metrics. To quantitatively characterize the formalization overhead and intrinsic difficulty of each proposition, we define the following four indicators. *(1) Effective Code Scale* L_{eff}. As the most intuitive empirical measure of formalization workload, we adopt effective lines of code. Through static analysis, after automatically stripping single-line and nested block comments from the Lean 4 source code, blank lines are excluded, yielding L_{eff}. *(2) Mathlib Coverage Ratio* C_{m}. Using Claude Opus 4.7, we evaluate the extent to which the required definitions and lemmas are already covered by the existing Mathlib. The model outputs a coverage ratio $C_{\text{m}} \in [0, 1]$, where higher values indicate greater reliance on pre-existing formalized components. *(3) Formalization Engineering Difficulty* D_{f}. From the same Claude Opus 4.7 evaluation, we obtain a difficulty score $D_{\text{f}} \in [0, 1]$ that reflects the engineering barriers at the code implementation level, including the complexity of key definitions and core lemmas. *(4) Mathematical Depth* D_{m}. Leveraging Gemini 3.1 Pro Preview, we quantify the intrinsic pure-mathematical challenge of each proposition, encompassing its underlying mathematical structure, proof bottlenecks, and prerequisite concepts, producing a mathematical difficulty score $D_{\text{m}} \in [0, 1]$.

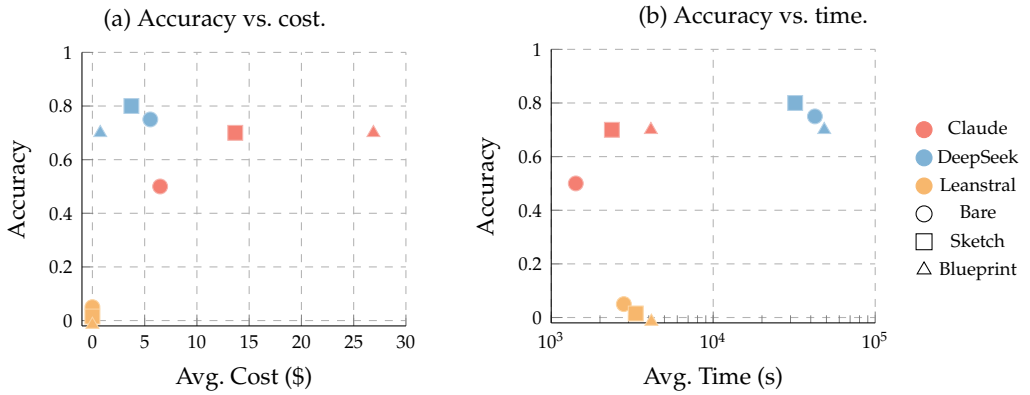


Figure 3: Comparison of three LLM-driven agents across three evaluation modes on 20 problems. Leanstral points (lower left) are slightly jittered along x for visibility.

Assembly Difficulty Score. To comprehensively measure the overall challenge of a proposition, we integrate the three evaluations into a single assembly difficulty score:

$$D = D_{\mathrm{f}} + D_{\mathrm{m}} + (1 - C_{\mathrm{m}}).$$

Here, the term $(1 - C_{\mathrm{m}})$ serves as a penalty: a lower Mathlib coverage ratio incurs a larger penalty, reflecting the increased engineering burden of manually formalizing missing foundational dependencies. We partition all candidate propositions into three tiers based on D, using split thresholds at 1.5 and 1.8 (see Table 2), and select approximately 6 to 8 problems from each tier to ensure balanced coverage across the full difficulty spectrum.

Mathlib Coverage Extremes. To capture varying degrees of dependency on existing formalization infrastructure, we stratify by C_{m} and enforce quotas for two extreme cases: (1) problems with $C_{\mathrm{m}} < 0.3$, which require substantial manual development of missing foundational components; (2) problems with $C_{\mathrm{m}} > 0.7$, which primarily exercise the ability to effectively reuse standard libraries.

Effective Code Scale. To avoid over-selecting either trivial small proofs or excessively large formalizations, we stratify by L_{eff}. All candidates are divided into three tiers according to L_{eff}: short proofs (<300 lines, blue), medium proofs (300–1000 lines, green), and long proofs (>1000 lines, yellow). We require at least three problems for each category, ensuring that the final subset includes short, medium, and large formalization tasks. Furthermore, we include several problems that currently lack formal proofs (red) to fully explore the boundaries of the models' capabilities.

Based on the above criteria, we perform stratified filtering to ultimately select 20 problems that constitute our test subset. The detailed statistical distribution of the selected subset is presented in Table 2.

5.2 Experiment Results

To ensure a fair and comprehensive comparison of agentic proving capabilities across different models, we develop a unified evaluation framework for the *Live Lean Triathlon*. This framework upgrades the original Numina-Lean-Agent [10] by migrating the MCP tool protocol to a highly adaptable command-line interface (CLI) invocation mechanism, thereby achieving broader and more flexible compatibility with various base models.

Under this unified framework, we instantiate and evaluate three representative base models: Claude Opus 4.7 (Claude), DeepSeek v4 Pro (DeepSeek), and LeanStral. All models were tested on the 20-theorem subset detailed in Section 5.3, comprehensively spanning the three distinct task modes.

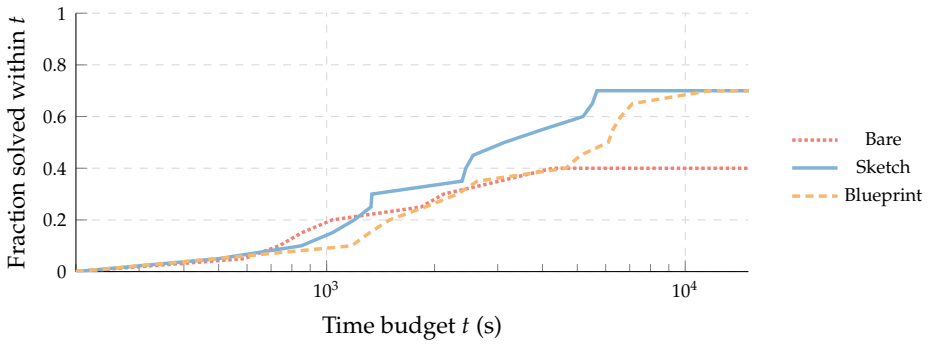


Figure 4: Cumulative success rate vs. time budget of Claude on the selected subset. Sketch-Guided reaches a 0.7 success rate fastest and most efficiently, while Blueprint-Guided achieves the same rate at much higher cost. Bare-Statement plateaus at 0.4, highlighting the necessity of structural guidance.

Figure 3 reports the accuracy, inference cost and time expenditure for each model across different modes. Synthesizing these results, we derive several key findings regarding the models' problem-solving paradigms. Under *Sketch-Guided* and *Blueprint-Guided* modes, DeepSeek and Claude achieved nearly identical problem-solving success rates , while DeepSeek incurred significantly lower costs (Figure 3a), albeit with a trade-off of substantially higher time expenditure (Figure 3b).

Furthermore, at the model level, DeepSeek and Claude exhibit distinctly different capability traits. As shown in Figure 3b, Claude tends toward rapid response but premature abandonment: its single-step inference is exceptionally fast, leading to a rapid initial climb in the curve; however, when faced with complex proof paths or continuous compilation errors, it lacks the autonomy for self-correction and restructuring. In contrast, DeepSeek demonstrates strong exploration resilience: although its search process is slow and highly time-consuming, it persistently navigates the massive state space. This qualitative difference is particularly evident in the unassisted *Bare-Statement* mode: Claude saturates early and suffers a significant drop in performance, whereas DeepSeek exhibits a striking late-stage advantage, ultimately surpassing Claude's *Sketch-Guided* performance in the absolute number of solved problems. This clearly demonstrates that DeepSeek retains robust autonomous planning capabilities even in a "zero-scaffolding" environment. Finally, although the smaller-parameter Leanstral solves very few complete problems, it still successfully fills in some local `sorry` sub-goals, indicating that it possesses basic low-level tactic execution abilities but remains significantly deficient in macroscopic proof planning.

5.3 Analysis of Different modes

As shown in Figure 4, the formal sketch (*Sketch-Guided*) yields the most immediate and efficient gains, reaching a saturation plateau of 0.7 significantly earlier than the blueprint mode. Conversely, while the informal blueprint (*Blueprint-Guided*) eventually matches this 0.7 success rate, its trajectory demands substantially higher inference time and computational cost. This disparity suggests that structured prompts directly aligned with the target language spare the prover the translation burden of auto-formalization, making sketches the most cost-effective approach at present. Meanwhile, the unguided *Bare-Statement* mode plateaus prematurely at 0.4, exposing the strict limitations of autonomous search without structural guidance.

Nevertheless, informal proofs remain highly valuable, as they preserve rich human mathematical intuition and high-level reasoning that can inspire alternative proof strategies as cross-modal alignment and long-context capabilities improve. More importantly, this striking performance variance validates the necessity of the multi-mode "Triathlon" framework. Theorem proving configurations test fundamentally distinct capabilities: *Bare-Statement* assesses zero-shot autonomous search; *Sketch-Guided* evaluates micro-level proof decomposition and library retrieval; and *Blueprint-Guided* requires a deep integration of

auto-formalization and path search. Our results reveal that a model's proficiency in one dimension (e.g., sketch-guided inference) cannot be linearly extrapolated to others (e.g., exploration from scratch). Therefore, only a multi-mode cross-evaluation can accurately map the complete capability boundaries of modern LLMs in formal mathematics.

6 Future work

A primary objective of this work is to establish a benchmark that rigorously assesses the capabilities of AI models across practically relevant use cases. To this end, we have incorporated both informal proofs and formal statements spanning a diverse range of mathematical fields. Moving forward, several promising avenues exist to further enhance the dataset's utility and scope.

While our current evaluation focuses exclusively on formal proof generation, real-world applications often demand a tighter integration of formal and informal mathematics. By defining a taxonomy of "states of knowledge" surrounding a theorem (ranging from unstated hypotheses to complete formal proofs), we can map out a broader spectrum of advanced AI tasks. Specifically, future iterations of this benchmark could encompass: (1) Statement Auto-Formalization: Translating informal statements into formal equivalents, evaluated by requiring the model to prove equivalence to the ground truth under succinctness constraints to avoid trivialization. (2) Informalization: Generating natural language explanations from formal proofs, verified via human evaluation for clarity and correctness. (3) Proof Sketching [7]: Producing informal sketches to guide formal proof construction, which can be automatically evaluated by using the sketch to condition a downstream auto-formalization model. Despite their verification challenges, these tasks reflect key formalization use cases for AI agents, prioritizing their inclusion in future work.

References

[1] Justin Asher. Leanexplore: A search engine for lean 4 declarations, 2025. URL https://arxiv.org/abs/2506.11085.

[2] Associate Editors of Mathematical Reviews and zbMATH. MSC2020: Mathematics subject classification system. https://msc2020.org, 2020. Revised jointly by the editorial staffs of Mathematical Reviews and Zentralblatt für Mathematik.

[3] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. URL https://arxiv.org/abs/2302.12433.

[4] Oliver Dressler. Lean LSP MCP: Tools for agentic interaction with the Lean theorem prover. https://github.com/oOo0oOo/lean-lsp-mcp, March 2025.

[5] Moritz Firsching, Paul Lezeau, Salvatore Mercuri, Miklós Z. Horváth, Yaël Dillies, Calle Sönne, Eric Wieser, Fred Zhang, Thomas Hubert, Blaise Agüera y Arcas, and Pushmeet Kohli. Formal conjectures: An open and evolving benchmark for verified discovery in mathematics, 2026. URL https://arxiv.org/abs/2605.13171.

[6] Kurt Gödel. Letter to John von Neumann, March 20, 1956. In Solomon Feferman, John W. Dawson, Warren Goldfarb, Charles Parsons, and Wilfried Sieg (eds.), *Kurt Gödel: Collected Works, Volume V: Correspondence H–Z*, pp. 373–376. Oxford University Press, 2003. Originally written March 20, 1956.

[7] Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

[8] Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numina-math. Hugging Face repository, 2024. URL https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf.

[9] Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, Zhengfeng Yang, Jujian Zhang, Lihong Zhi, Jia Li, and Zhengying Liu. Combibench: Benchmarking llm capability for combinatorial mathematics, 2025. URL https://arxiv.org/abs/2505.03171.

[10] 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, 2026. URL https://arxiv.org/abs/2601.14027.

[11] Alex Meiburg. Quantum information in lean. https://github.com/Timeroot/Lean-QuantumInfo, 2024.

[12] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe (eds.), *Automated Deduction – CADE 28*, pp. 625–635, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5.

[13] Azim Ospanov, Farzan Farnia, and Roozbeh Mohit. minif2f-lean revisited: Reviewing limitations and charting a path forward. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 79964–80002. Curran Associates, Inc., 2025.

[14] Alec Radford, Jeffrey Wu, Rewon Ext Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[15] Nikil Ravi, Kexing Ying, Vasilii Nesterov, Rayan Krishnan, Elif Uskuplu, Bingyu Xia, Janitha Aswedige, and Langston Nashold. Formalproofbench: Can models write graduate level math proofs that are formally verified? In *ICLR 2026 Workshop: VerifAI-2: The Second Workshop on AI Verification in the Wild*, 2026. URL https://openreview.net/forum?id=1LdnVndCEG.

[16] Marco Dos Santos, Hugues de Saxcé, Haiming Wang, Ran Wang, Mantas Baksys, Mert Unsal, Junqi Liu, Zhengying Liu, and Jia Li. Kimina lean server: A high-performance lean server for large-scale verification. *arXiv preprint arXiv:2504.21230*, 2025.

[17] The Lean Community. 1000+ theorems in Lean. https://1000-plus.github.io/, 2024. Accessed: 2026.

[18] The Lean Prover Team. Comparator. https://github.com/leanprover/comparator, 2025.

[19] The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pp. 367–381, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370974. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

[20] The Stacks Project Authors. The Stacks Project. https://stacks.math.columbia.edu/, 2026. Accessed: 2026.

[21] Joseph Tooby-Smith. Heplean: Digitalising high energy physics. *Computer Physics Communications*, 308:109457, 2025. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2024.109457. URL https://www.sciencedirect.com/science/article/pii/S0010465524003801.

[22] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11354.

[23] Rongge Xu, Hui Dai, Yiming Fu, Jiedong Jiang, Tianjiao Nie, Junkai Wang, Holiverse Yang, and Zhi-Hao Zhang. Leancat: A benchmark suite for formal category theory in lean (part i: 1-categories), 2026. URL https://arxiv.org/abs/2512.24796.

[24] Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Zheng Yuan, Huajian Xin, Wenhao Huang, Yandong Wen, Ge Zhang, and Weiyang Liu. Formalmath: Benchmarking formal mathematical reasoning of large language models, 2025. URL https://arxiv.org/abs/2505.02735.

[25] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2022. URL https://arxiv.org/abs/2109.00110.

A Implementation Details of Experiments

A.1 Prompts Used in the Experiments

A.1.1 Model evaluation

```
**FIRST ACTION**: Read the skill overviews now | `skills/search/SKILL.md`,
↪  `skills/verification/SKILL.md`, `skills/llm/SKILL.md`,
↪  `skills/code-transform/SKILL.md`, `skills/sorrifier/SKILL.md`. For any
↪  tool's exact CLI invocation, read the corresponding `reference-<tool>.md`
↪  in that directory.

You are Lean Coordinator.

## MUST READ FIRST
(Paths are relative to cwd `numina-lean-agent/`; docs live under
↪  `prompts/docs/prompts/`.)
1. prompts/docs/prompts/common.md (shared rules)
2. prompts/docs/prompts/coordinator.md (your workflow)
3. prompts/docs/prompts/blueprint_agent.md (blueprint creation and refinement)
4. prompts/docs/prompts/proof_agent.md (proof strategies)
5. prompts/docs/prompts/sketch_agent.md (formalization)
6. GPT_HINT.md (hints for the proof, if available)

## Key Rules (see docs for details)
- ALL work must go through Task tool subagent. You are forbidden from doing work
↪  directly.
- Use `python skills/cli/lean_check.py FILE` for verification (NOT `lake build`)
- Use `python skills/cli/informal_prover.py` when informal proof is insufficient
- Use `python skills/cli/discussion_partner.py` for strategic advice from Gemini
- Mathlib not having it is NOT an excuse to give up - build it yourself step by
↪  step
- Use emoji for status: done, partial, todo
- Tmp file: First add comment in original file, THEN create tmp file in `tmp/`
↪  subfolder alongside original (NEVER in /tmp)
- Don't use axiom. Use sorry for unproven statements.
- DO NOT say 'Mathlib lacks in some infrastructure'. You are powerful, you can
↪  create a new file and implement the missing infrastructure.
- DO NOT do enumerate when there are infinite possibilities. You should think
↪  critically and creatively or discuss with Gemini to find a general solution.

## Workflow
1. Read target file to understand current state
2. For each lemma needing work:
   - If informal proof unclear → Sketch Agent (use `python
   ↪  skills/cli/informal_prover.py`)
   - If formalized but unproven → Proof Agent
   - If too complex/stuck → Blueprint Agent (split into sub-lemmas)
   - If proof broken and needs isolation → use **sorrifier** workflow
   ↪  (`skills/sorrifier/SKILL.md`)
3. Update BLUEPRINT immediately after progress

## HINT
There is a hint for the proof in the GPT_HINT_v1.md file. Please read it
↪  carefully. You can mention it in your prompt for subagents.

## Blueprint (authoritative informal proof) [Only contain for Blueprint-Guided
↪  Mode]

Read the blueprint file at:
```

```
  __BLUEPRINT_PATH__

It is the authoritative informal proof for this theorem. Use it to guide the
formal proof: each `\label{lem:...}` entry corresponds to a Lean lemma name
given by `\lean{...}`, and the `\uses{...}` lines define the dependency graph
between lemmas. Do not change the statements of theorems or lemmas.

## Target

The target folder is:

  __TARGET_FOLDER__

It contains `MainTheorem.lean` (the goal; statements must remain unchanged)
and `BackgroundLemmas.lean` (helpers, currently with `sorry` placeholders).
Prove every `sorry` and make the folder compile cleanly.

## Progress tracker (BLUEPRINT.md)

Maintain a `BLUEPRINT.md` file **inside the target folder above**
(`__TARGET_FOLDER__/BLUEPRINT.md`) as the single-source-of-truth progress
tracker. If it does not exist, **copy the template from
`prompts/BLUEPRINT_template.md`** and fill it in using the `.tex` blueprint:
one entry per `\label{lem:...}` / `\label{thm:...}`, in dependency order,
with `lean:` / `file:` / `uses:` / `status:` / `attempts:` populated.
Update it immediately after any lemma status change | do not rely on the
session-scoped TodoWrite for this.

## Session End (CRITICAL FORMAT)

**Your response MUST end with exactly this line (no markdown, no extra text):**

```
END_REASON:COMPLETE
```
or
```
END_REASON:LIMIT
```

- `COMPLETE`: Main theorem proven successfully
- `LIMIT`: Made progress but can't complete in this session

This line must be the LAST line of your response. No text after it.
```

A.1.2 Mathlib Coverage and Formalization Engineering Difficulty Evaluation

```
# SYSTEM
You are a Lean 4 formalization expert. You evaluate mathematical theorems for
↪  their formalizability in Lean 4 using Mathlib.
Always respond with valid JSON only | no markdown fences, no extra text.

# USER
Analyze the following Lean 4 formalization problem from the perspective of a
↪  formalization architect.
These problems are typically well-known mathematical theorems (e.g., from
↪  Wikipedia) and represent significant formalization efforts.

## Problem Name
```

```
{name}

## Lean 4 Code
```lean4
{code}
```

Task 1: Engineering Analysis
Instead of just guessing a score, analyze the *engineering effort* required to formalize it:
1. What mathematical objects/structures need to be defined?
2. What are the key intermediate lemmas required?
3. Which parts are likely already in Mathlib, and which parts must be built from scratch?

Task 2: Mathlib Coverage Assessment
Estimate what percentage of the required mathematical machinery is already available in Mathlib.

Task 3: Formalization Difficulty Scoring Rubric (1-10)
Based on your analysis in Task 1, assign a `difficulty_score` strictly using this 3-tier rubric:
- 1-4 (Standard Advanced Theorem): Requires 100 to 1,000 lines of code. Involves formalizing graduate-level concepts currently missing from Mathlib. Typeclass searches and defeq issues are challenging but solvable using standard Lean 4 architectural patterns. (Anchors: Sylow's Theorems, basic Galois Theory, standard Lebesgue integration proofs).
- 5-7 (Deep Infrastructure Project): Requires 1,000 to 5,000+ lines of code. Involves developing an entire new mathematical subfield from scratch. Navigates extreme Mathlib hierarchy complexities (e.g., advanced schemes, spectral sequences, complex manifolds). The main bottleneck is translating highly abstract math into strict dependent types without causing severe performance issues. (Anchors: Prime Number Theorem, Independence of Continuum Hypothesis).
- 8-10 (The Impenetrable Fortress / Mega-Project): STRICTLY reserved for multi-year, multi-author mega-projects that push Lean to its absolute breaking point. Requires >30,000 lines of code, heavy custom metaprogramming (macros/tactics), or fundamentally patching Lean's foundations. (Anchors: Liquid Tensor Experiment, Sphere Eversion, Fermat's Last Theorem). DO NOT assign 8-10 unless it is a literal historic milestone of this magnitude.

Respond with EXACTLY this JSON structure:
{{
  "mathlib_coverage": {{
    "analysis": "<brief analysis of whether required math is in Mathlib>",
    "covered": <true or false>,
    "coverage_pct": <integer 0-100>
  }},
  "formalization_analysis": {{
    "required_definitions": ["List 1-3 missing or complex mathematical definitions needed"],
    "key_lemmas_needed": ["List 1-3 crucial intermediate lemmas to prove the main theorem"],
    "mathlib_gap": "<Analyze what specific machinery is missing or hard to use in Mathlib for this problem>",
    "tactic_complexity": "<E.g., Requires heavy use of 'aesop', complex custom induction, or tricky rewrites>"
  }},
  "formalization_effort": {{
```

```
    "estimated_lines_of_code": <integer estimate of how many lines the full
    ↪  proof would take>,
    "difficulty_score": <integer 1-10 based STRICTLY on the 3-tier rubric>,
    "justification": "<One sentence explicitly matching your analysis to a
    ↪  specific tier in the rubric>"
  }}
}}
```

A.1.3 Mathematical Depth Evaluation

```
You are a mathematics professor analyzing a well-known theorem or complex
↪  problem.

## Problem Name
{name}

## Lean 4 Code
```lean4
{code}
```

Task 1: Mathematical Structure Analysis
Analyze the fundamental mathematical structure of the theorem and the logical
↪ steps required to prove it. Identify the conceptual bottleneck that makes
↪ this theorem profound or difficult to prove rigorously.

Task 2: Mathematical Difficulty Scoring Rubric (1-10)
Based on your analysis in Task 1, assign a `score` strictly using this 3-tier
↪ rubric:
- 1-3 (Advanced Undergraduate): Core theorems from advanced undergraduate
↪ courses (e.g., Real/Complex Analysis, Galois Theory, Point-set Topology).
- 4-7 (Graduate Level): Material typically found in graduate courses. Requires
↪ deep theoretical insight, multi-step logical reasoning across different
↪ mathematical domains (e.g., Algebraic Geometry, advanced Measure Theory).
- 8-10 (Research / Historic Level): Extremely deep theorems, historic
↪ milestones, or open conjectures (e.g., Fermat's Last Theorem, Prime Number
↪ Theorem, complex modern classifications).

Respond with EXACTLY this JSON (no markdown fences, no extra text):
{{
  "mathematical_analysis": {{
    "proof_strategy": "<Describe the standard mathematical strategy used to
    ↪ prove this theorem>",
    "key_conceptual_bottleneck": "<What is the single hardest logical leap or
    ↪ most non-trivial construction in the proof?>",
    "required_prerequisites": ["List 2-3 advanced mathematical concepts required
    ↪ to even understand the proof"]
  }},
  "math_difficulty": {{
    "score": <integer 1-10 based STRICTLY on the 3-tier rubric>,
    "justification": "<One sentence explicitly explaining why the conceptual
    ↪ bottleneck places it in this specific rubric tier>"
  }},
  "category": "<e.g., Algebraic Geometry, Analytic Number Theory, Algebraic
  ↪ Topology, Model Theory, etc.>"
}}
```