ITPEVAL: Benchmarking Formal Translation Across Interactive Theorem Provers

Jiayi Wu [1] **Robert Joseph George** [2] **Anima Anandkumar** [2]

Abstract

Formal theorem proving has emerged as a frontier challenge for machine learning, yet the ecosystem is fragmented: proofs remain siloed across incompatible systems, limiting both training data for learning-based provers and the portability of verified results. We present **ITPEVAL**, the first benchmark for evaluating automated formal proof translation across four major ITPs (Lean 4, Rocq, Isabelle, and HOL Light), spanning two distinct logical foundations. Our benchmark comprises 1,560 source files and 6,848 theorems organized into a *controlled* tier of axiomatized files that isolates foundational translation difficulty, and an *ecosystem* tier drawn from real libraries that exposes API and proof-style mismatches. We release `itpeval`, a unified multi-ITP verification infrastructure with state-isolated warm backends that preserve per-artifact native checking semantics. We evaluate both *statement* and *proof* translation across five frontier and open-weight LLMs on 12 directed translation pairs: statement translation peaks at 29.1% pass@1 and proof translation at 10.5%; controlled theorems reach 29.7% proof pass@1 versus 5.2% for ecosystem-level translations, confirming that library mismatch is the dominant bottleneck. In addition to pass@k evaluation, a deterministic Lean 4 BEq check establishes equivalence for 54.0% of verified source $\mapsto$ Lean 4 miniF2F statement translations, showing that native type-checking alone can substantially overestimate semantic fidelity; in an autoformalization/auto-informalization round-trip study, Rocq and HOL Light are easier formalization targets than Lean 4 and Isabelle, while multi-ITP context improves pooled Lean 4 success from 4.8% to 10.6%. Our benchmark[1], verification infrastructure, and evaluation pipelines[2] are publicly released.

[1]Brown University [2]California Institute of Technology. Correspondence to: Jiayi Wu <jiayi_wu4@brown.edu>.
The 3rd AI for Math Workshop at the 43rd International Conference on Machine Learning (ICML), 2026. Copyright 2026 by the author(s).

[1]Data: `anonymous-itp/itpeval`.

1. Introduction

Formal mathematics has become an increasingly important domain for AI research: large language models now achieve strong results on automated theorem proving (Polu et al., 2022; Yang et al., 2023; First et al., 2023; Xin et al., 2024), autoformalization (Wu et al., 2022; Jiang et al., 2023b), and mathematical reasoning more broadly. Yet while proving and autoformalization have been extensively researched and benchmarked, the formal ecosystem remains fragmented: each interactive theorem prover (ITP) implements its own logical foundation, tactic language, and mathematical library, and *interoperability* between systems has been largely overlooked.

This fragmentation has concrete consequences: a theorem proved in Lean's Mathlib (mathlib Community, 2020) cannot be invoked in Isabelle's Archive of Formal Proofs (Archive of Formal Proofs, 2026), even when both encode the same mathematics. Cross-ITP translation, the task of converting formal proofs between systems while preserving correctness, would address this in two ways. First, it would let verified results transfer across ITP ecosystems, reducing duplicate formalization effort and expanding the training data available to learning-based provers. Second, it would inform autoformalization and auto-informalization, since both settings require preserving theorem meaning across different surface languages and libraries. We therefore also evaluate a round-trip autoformalization/auto-informalization setting in which multi-ITP context is used during generation.

Despite this importance, cross-ITP translation has received little systematic study. While formalization tracking efforts such as Formalizing 100 Theorems (Wiedijk, 2008) and the 1000+ Theorems project (1000+ Theorems Contributors, 2026) catalog which classical results have been proved in which provers, and logical frameworks such as Dedukti (Assaf et al., 2023; Dowek & Thiré, 2023) enable proof exchange via shared intermediate representations,

[2]Code: `lean-dojo/ITPEval`.

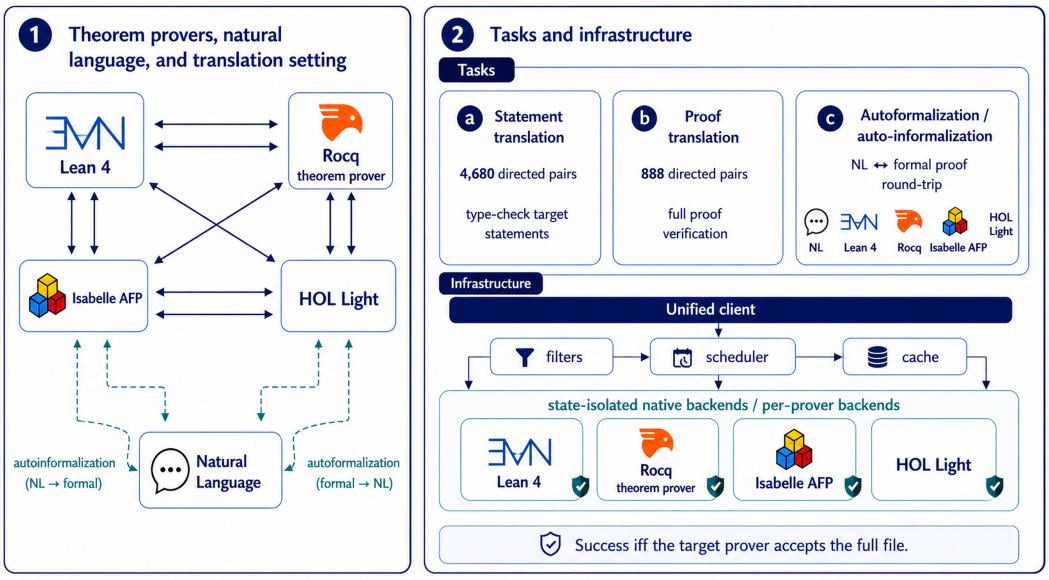

Figure 1. **Overview of ITPEval.** The figure summarizes the cross-ITP translation setting, natural-language round-trip tasks, and the unified state-isolated verification infrastructure.

none of these provide evaluation benchmarks for measuring translation quality. Without standardized benchmarks, it is difficult to measure whether LLM-based or symbolic translation methods produce correct, idiomatic output, or to identify which system pairs and proof features remain out of reach.

Constructing such a benchmark is itself non-trivial: each ITP community has developed its library largely independently, so no large aligned corpus of proofs exists across systems, and even theorems that have been formalized in multiple ITPs typically differ in proof structure, lemma dependencies, and library APIs. The only existing benchmark is Babel-formal (Stoskopf et al., 2025), limited to 14 aligned files between Lean and Rocq. Evaluation methodology is itself an open problem: type-checking alone has a high false-positive rate for semantic correctness (Poiroux et al., 2025; Liu et al., 2025b), and code translation benchmarks (Yan et al., 2023; Zheng et al., 2024) usually apply only partially since ITP translation must navigate logical foundation differences, not just syntactic ones. These gaps motivate a benchmark that jointly addresses cross-system coverage and foundation-aware evaluation.

We present **ITPEVAL**, a benchmark and unified multi-ITP verification infrastructure for studying formal translation across major theorem-prover ecosystems. ITPEval spans Lean 4, Rocq, Isabelle, and HOL Light, covering both dependent type theory and higher-order logic. Its design separates two sources of difficulty: a *controlled* tier of self-contained, axiomatized files that isolates foundational translation issues, and an *ecosystem* tier of real library formalizations that exposes API mismatches, naming conventions, automation behavior, and proof-style differences. This tiered structure quantifies the additional cost of library and proof-engineering dependencies beyond foundation-level translation. Our contributions are:

- **A four-way aligned cross-ITP benchmark** with 1,560 source files and 6,848 theorems and lemmas across Lean 4, Rocq, Isabelle, and HOL Light. The benchmark yields 12 directed translation pairs and is organized into a controlled tier (64 axiomatized files, 660 lemmas) and an ecosystem tier (232 Formalizing 100 Theorems files and 1,264 miniF2F statements).
- **A unified multi-ITP client and verification infrastructure** that exposes heterogeneous theorem provers through a common evaluation interface with environment-keyed scheduling, state-isolated warm backends, and per-artifact diagnostics.
- **A systematic evaluation of closed-source API and open-weight LLMs** on statement and proof translation. We evaluate GPT-5.5, Claude Sonnet 4.6, Gemini 3.1 Pro, DeepSeek-V4-Pro, and Qwen3-235B-A22B, with every output checked by the native target prover rather than by surface-level heuristics.
- **An autoformalization and auto-informalization round-trip study** showing target-prover effects: Rocq and HOL Light are easier formalization targets, while multi-ITP context improves pooled Lean 4 success.

Table 1. **Positioning of ITPEVAL.** A = aligned data, B = benchmark, N = target-prover checking, U = unified verifier. ● = central or supported, ◆ = partial or limited, ❍ = not a focus.

Work	Systems	A	B	N	U	Role
Interoperability and overlap						
Formalizing 100 Theorems / 1000+ Theorems (Wiedijk, 2008; 1000+ Theorems Contributors, 2026)	Many ITPs	◆	❍	❍	❍	Overlap catalogues
Dedukti / Logipedia / EuroProofNet (Assaf et al., 2023; Dowek & Thiré, 2023; Thiré, 2020)	Many logics	◆	❍	◆	◆	Proof exchange
OpenTheory (Hurd, 2011)	HOL family	◆	❍	◆	◆	HOL packages
Formal translation and autoformalization						
Babel-formal (Stoskopf et al., 2025)	Lean, Rocq	●	●	●	❍	Two-prover benchmark
MiniF2F in Rocq (Viennot et al., 2025)	Lean, Isabelle, Rocq	◆	◆	●	❍	Rocq statement port
MMA (Jiang et al., 2023a)	Lean, Isabelle	◆	❍	◆	❍	NL–formal data
ProofWala (Thakur et al., 2025a)	Lean, Coq/Rocq	◆	❍	●	◆	Proof search
KELPS (Zhang et al., 2025)	Lean, Coq/Rocq, Isabelle	◆	❍	◆	❍	Autoformalization
Interfaces and analogues						
LeanDojo / Pantograph / PISA / `itp-interface` (Yang et al., 2023; Aniva et al., 2025; Jiang et al., 2021; Thakur et al., 2025b)	One or two provers	❍	❍	◆	◆	Prover interfaces
CodeTransOcean / HumanEval-X / CRUXEval-X (Yan et al., 2023; Zheng et al., 2024; Xu et al., 2025)	Programming languages	●	●	◆	❍	Code translation
ITPEVAL	**Lean 4, Rocq, Isabelle, HOL Light**	●	●	●	●	**Four-way benchmark + verifier**

- **A Bidirectional Extended Definitional Equivalence check** (BEq) that tests whether verified translations preserve theorem meaning, implemented for Lean 4 targets. Applied to miniF2F statement translations, 174 of 322 verified source ↦ Lean 4 translations (54.0%) pass BEq; the remaining 46.0% expose either semantic drift or limitations of the restricted equivalence search.

2. Related Work

Formal systems. ITPEVAL spans two major proof-assistant traditions. Lean 4 (Moura & Ullrich, 2021) and Rocq (Bertot & Castran, 2010) are based on the Calculus of Inductive Constructions (CIC), while Isabelle/HOL (Nipkow et al., 2002) and HOL Light (Harrison, 2009) are based on Higher-Order Logic (HOL). Translation difficulty is not determined by foundation alone: tactic languages, proof idioms, libraries, automation, and naming conventions differ even within the same logical family.

Prior work. Prior work has tracked theorem coverage across provers and built proof-exchange infrastructure, but has not provided a balanced four-way benchmark for cross-ITP translation. Formalization trackers record overlap across systems (Wiedijk, 2008; 1000+ Theorems Contributors, 2026), while Dedukti, Logipedia, EuroProofNet, and OpenTheory provide proof exchange infrastructure (Assaf et al., 2023; Dowek & Thiré, 2023; Thiré, 2020; Hurd, 2011). Babel-formal (Stoskopf et al., 2025) is the closest prior benchmark, but covers Lean and Rocq only. Recent multilingual autoformalization and proof-search systems study natural language to formal translation or two-prover proof search, rather than balanced formal-to-formal translation across four ITPs (Jiang et al., 2023a; Thakur et al., 2025a; Zhang et al., 2025).

Evaluation interfaces. Target-prover checking is necessary for formal translation, but type checking alone does not guarantee semantic fidelity (Poiroux et al., 2025; Liu et al., 2025b). Existing interfaces such as LeanDojo, Pantograph, PISA, and `itp-interface` support interaction with individual provers or small prover sets (Yang et al., 2023; Aniva et al., 2025; Jiang et al., 2021; Thakur et al., 2025b). ITPEVAL instead contributes a unified, state-isolated multi-ITP verifier for benchmark-scale evaluation across Lean 4, Rocq, Isabelle, and HOL Light. Appendix D gives a more detailed comparison.

3. Benchmark Design

A benchmark for cross ITP translation must separate several sources of difficulty. Some failures come from differences in logical foundations, such as dependent type theory versus higher order logic. Others come from prover ecosystems: library APIs, naming conventions, automation, local helper lemmas, and proof idioms. ITPEVAL is designed to expose these effects separately, while evaluating both statement translation and full proof translation.

Tier A: controlled files. Tier A contains self-contained, axiomatized files. These files include their own definitions and assumptions, and avoid dependence on prover-specific libraries. They are intended to isolate foundational and language-level translation issues, such as type theory, quantification, implicit arguments, universe conventions, and proof syntax.

Tier B: ecosystem files. Tier B contains formalizations drawn from existing libraries and community developments. These files reflect real proof engineering practice: they use library lemmas, local helper definitions, tactic automation, naming conventions, and prover-specific mathematical structures. They therefore test the combined difficulty of translating both the mathematics and the surrounding ecosystem.

This two-tier design makes the main analysis interpretable. Performance on Tier A measures how well models handle controlled cross-system translation, while the gap between Tier A and Tier B measures the additional cost of real prover ecosystems.

3.1. Data Sources and Selection

A central design constraint is the **four-way intersection requirement**: every file in the benchmark must be formalized in all four ITPs. This ensures that every source–target pair is evaluated on exactly the same theorems, enabling clean directional and foundation-pair comparisons without missing-data confounds. We exclude files where a formal-

ization is unavailable in any of the four ITPs, where the source fails to compile in its native prover, or where theorem identity is ambiguous across systems.

Babel-formal (Tier A, 16 aligned items; 64 files; 660 lemmas). We adopt and extend the Babel-formal benchmark (Stoskopf et al., 2025) from its original 14-file Lean–Rocq scope to all four ITPs, re-formalizing each item in Isabelle and HOL Light and adding 2 new items formalized in all four systems from inception. Alignment is exact: we authored all four versions ourselves. Each item is a self-contained axiomatized theory (2–19 lemmas per prover-specific file) covering order theory, group theory, topology, and analysis. The axiomatized design isolates type-theoretic translation difficulty (dependent types, universe levels, implicit arguments) from library API mismatches.

Formalizing 100 Theorems (Tier B, 58 aligned items; 232 files; 4,924 lemmas). From Wiedijk's 100 classical theorems (Wiedijk, 2008), we retain the 58 with formalizations publicly available in all four ITPs, aligned by canonical list number (e.g., Theorem 1 = irrationality of $\sqrt{2}$). Because each prover's formalization was developed independently by its community, stylistic differences are genuine. Formalizations are sourced from Mathlib 4 (Lean 4), the standard library and MathComp (Rocq), the Archive of Formal Proofs (Isabelle 2024), and HOL Light's core theories. Each translation unit is the *entire source file*, including the headline theorem plus supporting lemmas and auxiliary definitions (mean 21 lemmas per file, range 1–95), reflecting realistic translation scenarios where proofs depend on locally-defined helpers.

MiniF2F (Tier B, 1,264 single-theorem files, statements only). The competition mathematics benchmark miniF2F (Zheng et al., 2022) provides formalized problem statements across Lean, Isabelle, and HOL Light; we extend coverage to Rocq and retain the 316 problems formalized in all four systems, aligned by standardized problem identifier (e.g., `amc12a_2008_p25`). Each file contains exactly one theorem statement. MiniF2F complements Formalizing 100 Theorems: its single-statement files test pure statement translation without multi-lemma structure, and its competition-math provenance provides diverse content (algebra, number theory, combinatorics) at controlled complexity.

3.2. Translation Modes

We evaluate two translation modes, with **statement translation as the primary task**. Statement translation is both more tractable and directly diagnostic of whether models can encode the same theorem across type-theoretic and definitional boundaries, without confounding this with proof-search difficulty.

Statement translation (`stmts`). Given a source file with all proof bodies replaced by placeholders (e.g., `sorry`), produce the corresponding file in the target ITP, also with placeholder proofs. Verification checks that the generated file *type-checks* in the target system; for multi-lemma files (Babel-formal, Formalizing 100 Theorems), all statements in the file must type-check together.

Proof translation (`proofs`). Given a complete proof file in the source ITP, produce a complete, compilable proof file in the target ITP. Verification requires that the entire generated file compiles without errors and with no sorry placeholders; for multi-lemma modules, every lemma must be fully proved.

3.3. Infrastructure

A substantial part of building ITPEVAL was designing a scalable verification infrastructure, not merely collecting aligned formalizations. Cross ITP evaluation is only meaningful if generated artifacts are checked by the target prover itself under conditions that faithfully reflect standalone verification. We refer to this as *native target-prover checking*. For this reason, all labels in ITPEVAL are produced by the actual target ITP. A translation is counted as *verified* only when the target prover accepts the entire generated file with exit code 0. Partial files, later declaration failures, and proof mode outputs containing placeholders are all counted as failures. For every artifact, the verifier records the final verdict, diagnostics, wall clock time, backend identity, and environment metadata.

The main systems challenge is that the target provers have very different execution models. Lean 4 and Rocq can be checked relatively directly, but Isabelle and HOL Light have much heavier startup and session costs. Isabelle verification is organized around sessions and theory files, while HOL Light runs through an OCaml-based environment whose initialization can dominate small checks. Treating each prover as a separate script would make evaluation fragile, difficult to restart, and hard to compare across targets. We therefore release `itpeval`, a unified multi-ITP client and toolchain manager. It installs and runs all four proof assistants behind a common `run_itp()` entry point: each target prover is materialized in its own task directory, while prover-specific details remain localized to adapters.

The central invariant is *state isolation*: every accelerated check must be observationally equivalent to verifying the generated artifact on its own. Figure 2 summarizes the verifier. Generated JSONL records pass through strict one-sided filters, content-addressed caching, environment keys, and cost-aware scheduling before entering a transactional `CheckRequest`/`CheckResult` API. The API then dispatches to prover-specific backends: Lean 4 checks fresh synthetic files, Rocq resets persistent REPL workers with

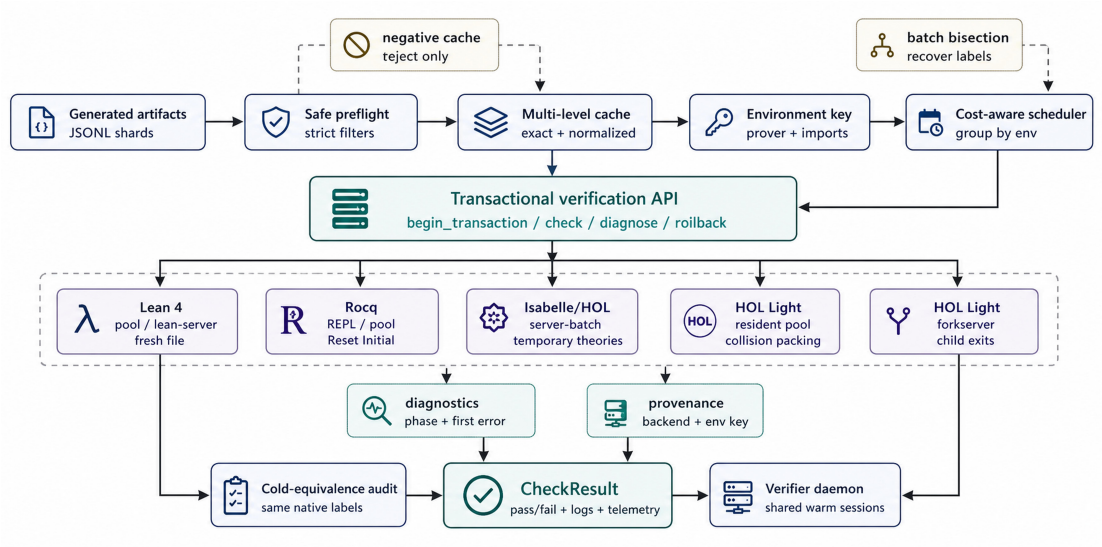

Figure 2. ITPEVAL verification infrastructure. Generated artifacts are filtered, cached, scheduled by environment, and checked by prover-specific warm backends while preserving native per-file checking semantics.

`Reset Initial.`, Isabelle checks temporary theories inside server sessions, and HOL Light uses collision-aware resident workers or a forkserver. Adaptive bisection recovers per-artifact labels after batch failures, append-only checkpoints make runs restartable, and telemetry records diagnostics, backend provenance, and timing. These mechanisms are part of the experimental protocol: without them, warm workers can leak declarations across files, batched checking can lose pass@1 labels, and target-prover effects can be confounded with verifier behavior.

3.4. Semantic Equivalence Checking

Target-prover type checking is necessary for correctness but not sufficient for semantic fidelity: a translated statement can type-check while expressing a weaker or shifted theorem (Poiroux et al., 2025; Liu et al., 2025b). To quantify this gap, we adapt *Bidirectional Extended Definitional Equivalence* (BEq), introduced by Liu et al. (2025a), to verified Lean 4 target translations. Given a generated Lean statement G and the reference Lean statement R, BEq checks both entailments $G \vdash R$ and $R \vdash G$; a translation passes BEq iff both directions succeed.

Each directional check constructs a fresh Lean file by renaming the assumed theorem to `stmt_assumed`, renaming the goal theorem to `beq_goal`, replacing the goal proof with restricted proof search, and running Lean. Liu et al.'s original implementation first tries `exact?` and then samples tactic sequences from an LLM under a restricted primitive set. We keep the same bidirectional criterion and restricted-primitive design, but replace LLM-sampled transformations with a deterministic Lean tactic cascade. This makes the metric reproducible and avoids adding a second model-dependent source of variance to the translation benchmark. Appendix F gives the full protocol and failure analysis.

4. Evaluation and Results

4.1. Evaluation Setup

We evaluate five zero-shot, greedy-decoded models ($T = 0$): GPT-5.5, Claude Sonnet 4.6, Gemini 3.1 Pro Preview, DeepSeek-V4-Pro, and Qwen3-235B-A22B. This set covers closed-source API systems and large open-weight models. Direct translation is the primary setting: given a source file and a target ITP, the model must emit only raw target ITP code, with no markdown fencing or explanation. We report pass@1 results.

4.2. Verification Throughput

The unified verifier is necessary in practice because target provers differ sharply in startup and checking cost. Table 2 reports wall clock time on 16 Babel-formal statement files. Lean 4 and Rocq already have modest cold checking costs, but still benefit from pooled or persistent workers. The largest gains occur for Isabelle and HOL Light, where startup and environment initialization dominate naive checking. Isabelle decreases from 472.2 s to 6.9 s using server batch sessions, while HOL Light decreases from 527.4 s to 149.0 s using resident or forkserver backends. These measurements show that scalable cross ITP evaluation requires infrastructure that is both faster and preserves per-file checking semantics.

Table 2. State preserving verifier backends and observed throughput on 16 Babel-formal statement files. Times are wall clock seconds; lower is better.

Prover	Backend	Reuse	Isolation	Cold	Warm
Lean 4	pool/server	compiler workers	fresh file	15.1	3.7
Rocq	REPL pool	persistent proc.	`Reset`	4.3	0.5
Isabelle	server batch	persistent session	temp. theories	472.2	6.9
HOL Light	forkserver	preloaded OCaml	child exit	527.4	149.0

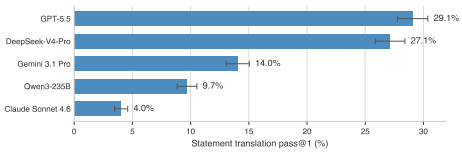

Figure 3. Statement translation pass@1 across all 4,680 directed file pairs. GPT-5.5 and DeepSeek-V4-Pro are the strongest overall, but even the best model succeeds on fewer than one third of pairs.

4.3. Statement Translation

Statement translation is the primary evaluation in ITPEVAL, assessed on all 4,680 directed pairs per model across the three benchmarks.

Figure 3 shows that statement translation remains challenging even before proof search is required. GPT-5.5 achieves the highest overall statement pass rate (29.1%), followed closely by DeepSeek-V4-Pro (27.1%). Gemini is lower at 14.0%, while Qwen3-235B-A22B and Claude Sonnet 4.6 remain below 10%. The gap between the two leading models and the rest suggests that statement translation depends on both formal syntax control and target library familiarity.

Table 3 shows that the controlled tier is consistently easier than the ecosystem tier. Babel-formal removes most library dependence and isolates foundational translation, while Formalizing 100 Theorems requires real library APIs, local helper lemmas, naming conventions, and prover-specific mathematical structures. MiniF2F is often easier than Formalizing 100 Theorems because it contains single theorem statements with fewer local dependencies. This suggests that ecosystem complexity, not mathematical difficulty alone, is the dominant source of failure.

4.3.1. Translation Direction Analysis

Translation direction reveals strong asymmetry. In statement mode, Isabelle and HOL Light are the easiest generation targets, whereas Lean 4 and Rocq remain difficult. The gaps are large: Lean 4 $\mapsto$ Isabelle reaches 17.6%, while Isabelle $\mapsto$ Lean 4 reaches only 4.3%; HOL Light $\mapsto$ Isabelle reaches 25.8%. These effects are not explained by logical foundation alone. CIC $\mapsto$ CIC directions are not consistently easier than cross-foundation directions, indicating that library conventions, elaboration behavior, and surface language matter more than the underlying foundation in many cases. Because every directed source–target pair is evaluated on the same mathematical content, these asymmetries reflect target-language generation difficulty rather than theorem selection. Full directional heatmaps are in Appendix A.

Table 3. Statement translation pass@1 by benchmark. The Tier A/B gap persists, but miniF2F outperforms Formalizing 100 Theorems despite containing competition mathematics.

Model	Babel (A, 192)	Formalizing 100 Theorems (B, 696)	miniF2F (B, 3792)
GPT-5.5	91 (47.4%)	106 (15.2%)	1163 (30.7%)
Gemini 3.1	63 (32.8%)	97 (13.9%)	496 (13.1%)
Claude 4.6	21 (10.9%)	32 (4.6%)	134 (3.5%)
DeepSeek-V4	33 (17.2%)	46 (6.6%)	1190 (31.4%)
Qwen3-235B	18 (9.4%)	10 (1.4%)	424 (11.2%)

Table 4. Proof translation pass@1 rates on 888 directed pairs. Tier A is consistently much easier than Tier B.

Model	Overall (888)	Babel (A, 192)	Formalizing 100 Theorems (B, 696)
GPT-5.5	**93 (10.5%)**	57 (29.7%)	36 (5.2%)
Gemini 3.1	45 (5.1%)	35 (18.2%)	10 (1.4%)
Claude 4.6	13 (1.5%)	11 (5.7%)	2 (0.3%)
DeepSeek-V4	13 (1.5%)	9 (4.7%)	4 (0.6%)
Qwen3-235B	5 (0.6%)	3 (1.6%)	2 (0.3%)

4.4. Proof Translation

We also evaluate **proof translation** on the 888 directed pairs from Babel-formal and Formalizing 100 Theorems.

Proof translation is much harder than statement translation. GPT-5.5 is the strongest model, but still solves only 10.5% of proof pairs overall. The tier gap is also sharper: GPT-5.5 reaches 29.7% on the controlled Babel-formal files but only 5.2% on Formalizing 100 Theorems. This indicates that proof failure is not just a matter of producing syntactically valid target code. Complete proof translation often requires recovering target library dependencies, local helper lemmas, automation behavior, and prover-specific proof idioms.

Direction reversal. Proof translation has a different target profile from statement translation. Isabelle is one of the easiest statement targets, but becomes the hardest proof target (1.9–3.2%). HOL Light remains comparatively strong, with proof pass rates between 4.3% and 5.7%. This suggests that well-formed statement generation and complete proof construction reward different target-language properties: Isabelle's structured Isar style is friendly for declarations, while complete proofs require more precise intermediate facts and automation control. HOL Light's compact OCaml tactic style appears more uniform for current models. Full per-model directional results are in Appendix A.

4.5. Statements vs. Proofs

On the 888 pairs where both modes are available, statement translation is clearly easier. A McNemar test on matched instances with the same theorem, direction, and model gives $\chi^2 = 249.19$ with $p < 0.0001$. The asymmetry is large: 417 instances pass only in statement mode, while 69 pass only in proof mode. The gap also varies by model. On the common benchmarks, GPT-5.5 drops from 22.2% on statements to 10.5% on proofs, Gemini from 18.0% to 5.1%, and Claude from 6.0% to 1.5%. At the theorem level, statement success is a near necessary condition for proof success: 32 theorems pass statements but never proofs, while no theorem passes proofs without any statement translation success. Thus, statement translation is a useful diagnostic for cross-system theorem encoding, but it is far from sufficient for full proof transfer.

4.6. Aggregate Analysis

We fit a logistic regression over all 27,840 evaluation records with binary verification outcome as the response (pseudo-$R^2 = 0.25$; full coefficients in Appendix B.6). The strongest predictors are target prover and benchmark identity. HOL Light and Isabelle are strong statement targets, while Formalizing 100 Theorems and miniF2F are harder than Babel-formal. Task mode is also important: proof translation has a large negative effect (-1.05). By contrast, sharing a logical foundation has only a weak positive effect ($+0.19$, $p = 0.001$). In proof mode, all four foundation pairs cluster between 2.2% and 4.5%, supporting the main conclusion that ecosystem mismatch dominates foundational mismatch once real prover libraries are involved.

Source complexity is another significant predictor. Longer source files are harder in proof mode (-0.54, $p < 0.001$), and source files associated with failed proof translations are longer than those associated with successful ones (median 3,992 vs. 1,978 characters). This is consistent with models struggling to translate long proofs into the target prover's local proof language, library names, and automation style.

4.7. Autoformalization and Auto-informalization

Type-checking confirms syntactic well-formedness but not semantic fidelity: a generated statement can type-check while expressing a weaker or shifted theorem. To probe whether models preserve mathematical content through the natural-language interface, we evaluate round trips that alternate between autoformalization (NL $\mapsto$ formal) and auto-informalization (formal $\mapsto$ NL). Each round trip has three steps: formalize a natural-language theorem statement into ITP code, informalize the generated code back to natural language, and re-formalize the informalized description. Verification at the first and third steps is a coarse executable signal of whether the round trip preserves a formalizable theorem.

Table 5. Multi-ITP autoformalization/auto-informalization round trips by target prover. Entries are pooled over five models; denominators exclude empty or skipped outputs. Rocq and HOL Light are easier targets at both formalization steps.

Target	Step 1 pass	Step 3 pass	Both pass
Lean 4	53/491 (10.8%)	56/493 (11.4%)	36
Isabelle	42/489 (8.6%)	21/490 (4.3%)	10
Rocq	152/465 (32.7%)	146/460 (31.7%)	101
HOL Light	144/460 (31.3%)	147/454 (32.4%)	103

We evaluate two configurations on 100 miniF2F theorems across all five models:

- **Single-ITP round trip:** NL $\mapsto$ Lean 4 $\mapsto$ NL $\mapsto$ Lean 4.
- **Multi-ITP round trip:** NL $\mapsto$ {Lean 4, Rocq, Isabelle, HOL Light} $\mapsto$ NL $\mapsto$ {Lean 4, Rocq, Isabelle, HOL Light}.

Table 5 shows that target prover choice dominates the round-trip outcome. Rocq and HOL Light verify about one third of pooled outputs at both formalization steps, roughly three times the Lean 4 rate and far above Isabelle in step 3. Their step-1 and step-3 rates are also stable, indicating that the same target differences persist after the natural-language intermediate step.

The benefit of multi-ITP context is model- and target-dependent. GPT-5.5 accounts for much of the HOL Light and Rocq success (91/100 and 61/100 at step 1; 92/100 and 62/100 at step 3), while Gemini is strongest on Rocq (61/75 at step 1; 57/67 at step 3) and relatively strong on Isabelle among non-empty outputs. Other model-target pairs show much smaller gains, so aligned multi-ITP context is useful evidence but not a uniform improvement.

For Lean 4 specifically, multi-ITP context improves pooled verification relative to the single-ITP Lean cycle: step 1 rises from 24/500 (4.8%) to 53/500 (10.6%), and step 3 rises from 39/500 (7.8%) to 56/500 (11.2%).

4.8. Semantic Equivalence Checking

We apply the BEq check (Section 3.4) to all 322 verified source $\mapsto$ Lean 4 statement translations on miniF2F.

Table 6 shows that 174 of 322 verified translations pass BEq (54.0%). This changes the interpretation of native verification: a Lean-accepted generated statement is often syntactically valid but not established equivalent to the reference by our restricted bidirectional check. Model differences are large. Claude Sonnet 4.6 has the highest BEq rate among verified translations (83.8%), while GPT-5.5, Gemini 3.1

Table 6. Semantic equivalence (BEq) of verified source ↦ Lean 4 statement translations on miniF2F. Pass% is the fraction of verified translations whose equivalence to the reference is established by BEq.

Model	Verified	BEq Pass	Pass%	BEq Fail
Claude Sonnet 4.6	74	62	83.8%	12
DeepSeek-V4-Pro	84	37	44.0%	47
Gemini 3.1 Pro	7	3	42.9%	4
GPT-5.5	29	10	34.5%	19
Qwen3-235B	128	62	48.4%	66
All models	322	174	54.0%	148

Pro, DeepSeek-V4-Pro, and Qwen3-235B range from 34.5% to 48.4%. BEq failures are mostly neither-direction failures (132 of 322 verified translations), while all asymmetric failures are backward-only (16 of 322): the reference entails the generated statement, but not conversely. This pattern is consistent with semantic weakening, though BEq failures should be read as *not established equivalent* rather than definitive semantic inequivalence because the deterministic cascade is incomplete.

Source direction has a smaller effect than model choice, but the rates are not identical: Rocq ↦ Lean passes at 52.4%, Isabelle ↦ Lean at 47.5%, and HOL Light ↦ Lean at 61.5%. HOL Light ↦ Lean has the highest BEq rate among verified translations, while Rocq ↦ Lean contributes the largest number of verified translations.

These results show that target-prover type checking is necessary but weak as a semantic metric. It remains the executable acceptance signal for translation, but equivalence-aware evaluation is needed to measure whether verified statements preserve theorem meaning.

4.9. Discussion

The main pattern is that ecosystem mismatch dominates foundational mismatch. Models perform much better on the controlled tier, where library dependence is removed, and drop sharply on ecosystem files that require real APIs, naming conventions, automation, and local helper lemmas. Future translation systems should therefore prioritize retrieval, library mapping, API alignment, and proof-style adaptation, not only logical foundation translation. This parallels findings in programming language translation, where library-centric errors dominate over core language failures (Xue et al., 2026).

Directional results reinforce this conclusion. If foundation similarity were decisive, Lean 4↔Rocq would be uniformly easiest. Instead, target effects dominate: Isabelle and HOL Light are strong statement targets, Isabelle becomes the hardest proof target, and HOL Light remains relatively strong for proofs. The four-way alignment is what makes this comparison meaningful, since every direction is evaluated on the same mathematical content.

Because miniF2F, Formalizing 100 Theorems, and Babel-formal are public resources, absolute pass rates may be affected by training-data contamination; miniF2F also already has cross-ITP ports. We therefore emphasize controlled comparisons across targets, directions, tiers, and verification criteria under a fixed evaluation protocol; future private or newly authored splits would strengthen claims about out-of-distribution generalization.

Finally, the infrastructure is part of the benchmark itself. Cross ITP results are only reliable when verifier state is isolated, labels are recovered per artifact, and backend provenance is recorded (Xin et al., 2026). The unified state-isolated client therefore fixes the unit of measurement to one generated artifact while making large-scale target-prover checking reproducible and auditable.

5. Conclusion

We presented **ITPEVAL**, a benchmark and unified verification infrastructure for formal translation across Lean 4, Rocq, Isabelle, and HOL Light. Across 1,560 source files, 6,848 theorems, and five LLMs, statement translation peaks at 29.1% pass@1 and proof translation at 10.5%. The controlled-to-ecosystem drop shows that library APIs, naming conventions, and automation dominate foundational mismatches.

A Bidirectional Extended Definitional Equivalence (BEq) check establishes equivalence for 54.0% of verified source ↦ Lean 4 miniF2F statement translations, so type-checking alone can overestimate semantic fidelity. The round-trip study shows that Rocq and HOL Light are easier autoformalization targets, while multi-ITP context improves pooled Lean 4 success. Limitations include zero-shot pass@1 evaluation and BEq coverage of Lean 4 targets only. Future work should add retrieval and repair, deploy BEq beyond Lean 4, and scale aligned corpora.

Acknowledgments

Jiayi Wu is supported by Caltech SURF fellowship. Robert Joseph George is supported by DARPA expMath and AA2.AIMING-1-NSF.2522494 (AIMing: Synergistic Advancement of AI and Mathematics with Dr. Sergei Gukov). Anima Anandkumar is supported in part by the Bren endowed chair, ONR MURI grant N00014-18-1-2624, DARPA expMath, and the AI2050 Senior Fellowship program at Schmidt Sciences.

References

1000+ Theorems Contributors. 1000+ theorems. https:

//1000-plus.github.io/, 2026.

Aniva, L., Sun, C., Miranda, B., Barrett, C., and Koyejo, S. Pantograph: A machine-to-machine interaction interface for advanced theorem proving, high level reasoning, and data extraction in lean 4, 2025. URL https://arxiv.org/abs/2410.16429.

Archive of Formal Proofs. Archive of formal proofs, 2026. URL https://www.isa-afp.org/. Edited by Manuel Eberl, Gerwin Klein, Peter Lammich, Andreas Lochbihler, Tobias Nipkow, Larry Paulson, René Thiemann, and Dmitriy Traytel. Accessed: 2026-05-05.

Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., and Saillard, R. Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory, 2023. URL https://arxiv.org/abs/2311.07185.

Bertot, Y. and Castran, P. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 3642058809.

Dowek, G. and Thiré, F. Logipedia: a multi-system encyclopedia of formal proofs, 2023. URL https://arxiv.org/abs/2305.00064.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. Baldur: Whole-proof generation and repair with large language models, 2023. URL https://arxiv.org/abs/2303.04910.

Harrison, J. Hol light: An overview. In Berghofer, S., Nipkow, T., Urban, C., and Wenzel, M. (eds.), *Theorem Proving in Higher Order Logics*, pp. 60–66, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03359-9.

Hurd, J. The opentheory standard theory library. In Bobaru, M., Havelund, K., Holzmann, G. J., and Joshi, R. (eds.), *NASA Formal Methods*, pp. 177–191, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-20398-5.

Jiang, A. Q., Li, W., Han, J. M., and Wu, Y. LISA: Language models of ISAbelle proofs. *6th Conference on Artificial Intelligence and Theorem Proving*, 2021. URL https://aitp-conference.org/2021/abstract/paper_17.pdf.

Jiang, A. Q., Li, W., and Jamnik, M. Multilingual mathematical autoformalization, 2023a. URL https://arxiv.org/abs/2311.03755.

Jiang, A. Q., Welleck, S., Zhou, J. P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., and Lample, G. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023b. URL https://arxiv.org/abs/2210.12283.

Li, Z., Wu, Y., Li, Z., Wei, X., Zhang, X., Yang, F., and Ma, X. Autoformalize mathematical statements by symbolic equivalence and semantic consistency, 2024. URL https://arxiv.org/abs/2410.20936.

Liu, Q., Zheng, X., Lu, X., Cao, Q., and Yan, J. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=hUb2At2DsQ.

Liu, X., Zhu, T., Dong, Z., Liu, Y., Guo, Q., Liu, Z., Chen, Y., and Luo, T. Assess: A semantic and structural evaluation framework for statement similarity, 2026. URL https://arxiv.org/abs/2509.22246.

Liu, Y., Zhu, T., Liu, X., Chen, Y., Liu, Z., Guo, Q., Zhang, J., Bao, K., and Luo, T. Generalized tree edit distance (gted): A faithful evaluation metric for statement autoformalization, 2025b. URL https://arxiv.org/abs/2507.07399.

Lu, J., Wan, Y., Huang, Y., Xiong, J., Liu, Z., and Guo, Z. Formalalign: Automated alignment evaluation for autoformalization, 2024. URL https://arxiv.org/abs/2410.10135.

mathlib Community, T. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, pp. 367–381. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

Mistral AI. Leanstral: Open-source foundation for trustworthy vibe-coding, 2026. URL https://mistral.ai/news/leanstral.

Moura, L. d. and Ullrich, S. The lean 4 theorem prover and programming language. In Platzer, A. and Sutcliffe, G. (eds.), *Automated Deduction – CADE 28*, pp. 625–635, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5.

Nipkow, T., Wenzel, M., and Paulson, L. C. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3540433767.

Poiroux, A., Weiss, G., Kunčak, V., and Bosselut, A. Reliable evaluation and benchmarks for statement autoformalization, 2025. URL https://arxiv.org/abs/2406.07222.

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning, 2022. URL https://arxiv.org/abs/2202.01344.

Stoskopf, T., Cohen, C., and Tabareau, N. Babel-formal: Translation of proofs between lean and rocq. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=WBI1PL0hZ2.

Thakur, A., Tsoukalas, G., Durrett, G., and Chaudhuri, S. Proofwala: Multilingual proof data synthesis and theorem-proving, 2025a. URL https://arxiv.org/abs/2502.04671.

Thakur, A., Tsoukalas, G., Durrett, G., and Chaudhuri, S. itp-interface: Generic interface for hooking up to any interactive theorem prover (itp) and collecting data for training ml models for ai in formal theorem proving, 2025b. URL https://github.com/trishullab/itp-interface.

Thiré, F. *Interoperability between proof systems using the logical framework Dedukti*. PhD thesis, 12 2020.

Viennot, J., Baudart, G., Arias, E. J. G., and Lelarge, M. Minif2f in rocq: Automatic translation between proof assistants – a case study, 2025. URL https://arxiv.org/abs/2503.04763.

Wiedijk, F. Formalizing 100 theorems. https://www.cs.ru.nl/~freek/100/, 2008.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models, 2022. URL https://arxiv.org/abs/2205.12615.

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024. URL https://arxiv.org/abs/2405.14333.

Xin, H., Li, L., Jin, X., Fleuriot, J., and Li, W. Ape-bench: Evaluating automated proof engineering for formal math libraries, 2026. URL https://arxiv.org/abs/2504.19110.

Xu, R., Cao, J., Lu, Y., Wen, M., Lin, H., Han, X., He, B., Cheung, S.-C., and Sun, L. Cruxeval-x: A benchmark for multilingual code reasoning, understanding and execution, 2025. URL https://arxiv.org/abs/2408.13001.

Xue, P., Zheng, K., Yang, Z., Pei, Y., Wu, L., Dong, J., Luo, X., Xiao, Y., Liu, F., Zhang, Y., Lyu, X., Li, X., Zhu, X., and Wang, C. Translibeval: Demystify large language models' capability in third-party library-targeted code translation, 2026. URL https://arxiv.org/abs/2509.12087.

Yan, W., Tian, Y., Li, Y., Chen, Q., and Wang, W. Codetransocean: A comprehensive multilingual benchmark for code translation, 2023. URL https://arxiv.org/abs/2310.04951.

Yang, K. and Deng, J. Learning to prove theorems via interacting with proof assistants, 2019. URL https://arxiv.org/abs/1905.09381.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models, 2023. URL https://arxiv.org/abs/2306.15626.

Zhang, J., Zhong, C., Xu, H., Li, Q., and Zhou, Y. Kelps: A framework for verified multi-language autoformalization via semantic-syntactic alignment, 2025. URL https://arxiv.org/abs/2507.08665.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2022. URL https://arxiv.org/abs/2109.00110.

Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Wang, Z., Shen, L., Wang, A., Li, Y., Su, T., Yang, Z., and Tang, J. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x, 2024. URL https://arxiv.org/abs/2303.17568.

Appendix Overview

Appendix guide

- **Appendix A: Full per-model results.** Proof, statement, HOL Light target behavior, and tier-gap breakdowns by model.
- **Appendix B: Statistical analysis.** Confidence intervals, model agreement, directional asymmetry, target rankings, foundation-pair effects, and variance decomposition.
- **Appendix C: Dataset and complexity characterization.** Corpus sizes, source lengths, long-context proof translation, theorem difficulty, and miniF2F categories.
- **Appendix D: Extended related work.** Formalization tracking, cross-ITP benchmarks, multilingual autoformalization, semantic fidelity, prover interfaces, and code-translation analogues.
- **Appendix E: Experimental details.** Round-trip autoformalization results, prompt templates, and the Babel-formal formalization process.
- **Appendix F: Semantic equivalence checking details.** BEq protocol, entailment asymmetries, and failure cases.
- **Appendix G: Infrastructure details.** State-isolated verification architecture, telemetry, reproducibility, and released artifacts.

A. Full Per-Model Results

A.1. Proof Translation

Figure 4 shows per-model proof translation pass@1 by source–target direction.

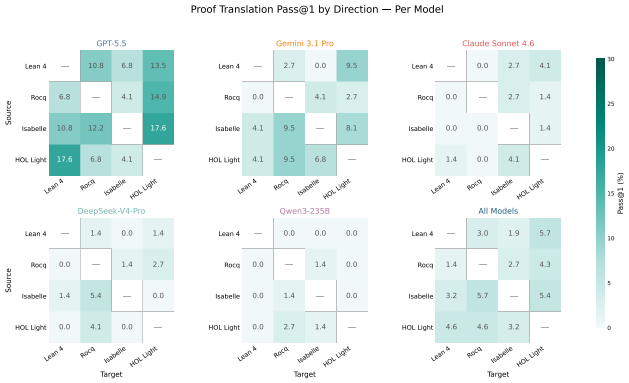

Figure 4. Proof translation pass@1 (%) by source–target direction, per model. GPT-5.5 dominates across most directions; DeepSeek-V4-Pro and Qwen3-235B show near-zero rates in many cells.

A.2. Statement Translation

Figure 5 shows per-model statement translation pass@1 by source–target direction.

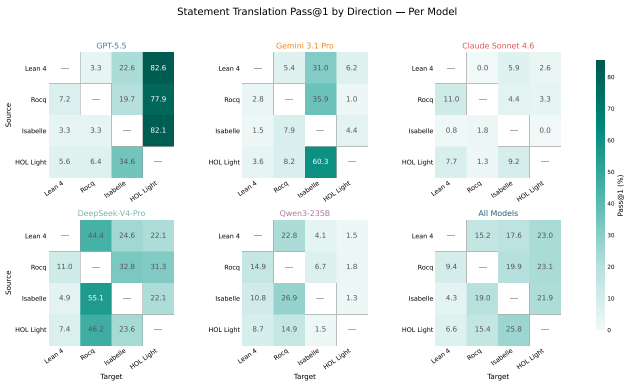

Figure 5. Statement translation pass@1 (%) by source–target direction, per model. Directional trends are broadly consistent but magnitudes vary: DeepSeek-V4-Pro achieves the highest per-cell rates while Claude Sonnet 4.6 is the weakest.

A.3. HOL Light as a Statement Target

HOL Light-target statement verification is an order of magnitude more expensive than other targets, and the results are highly model-dependent. Table 7 shows that GPT-5.5 achieves 80.9% on HOL Light-target statements and DeepSeek-V4-Pro reaches 25.1%, while the remaining three models stay below 4%. This gap suggests that HOL Light statement generation is particularly sensitive to decoding stability and syntax discipline, rather than exclusively to mathematical content.

Table 7. HOL Light-target statement translation by frontier model. Each direction has 390 pairs; Total has 1,170.

Model	L4 ↦ HL	Rocq ↦ HL	Isa ↦ HL	Total
GPT-5.5	322 (82.6%)	304 (77.9%)	320 (82.1%)	946 (80.9%)
Gemini 3.1	24 (6.2%)	4 (1.0%)	17 (4.4%)	45 (3.8%)
Claude 4.6	10 (2.6%)	13 (3.3%)	0 (0.0%)	23 (2.0%)

For Claude Sonnet 4.6, Isabelle ↦ HOL Light yields 0/390 passes, while Lean 4 ↦ HOL Light (2.6%) and Rocq ↦ HOL Light (3.3%) produce a handful. The dominant failure mode is syntax errors in HOL Light's OCaml-embedded tactic language: models frequently generate pseudo-HOL-Light that is syntactically close but fails the OCaml parser.

A.4. Tier Gap by Model

Figure 6 breaks down the Tier A (controlled) vs. Tier B (ecosystem) pass@1 gap for each model individually. The

tier gap is consistent across all five models in both statement and proof translation, confirming that ecosystem complexity—library APIs, naming conventions, automation behavior, and local helper lemmas—is the dominant source of difficulty beyond foundational translation issues.

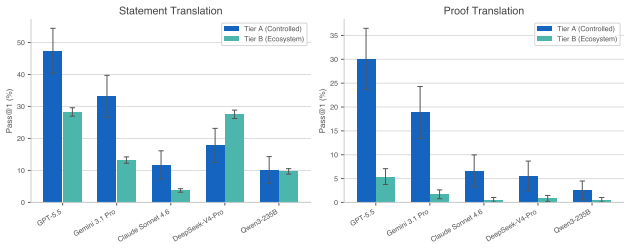

Figure 6. Tier A (controlled) vs. Tier B (ecosystem) pass@1 rates by model. Left: statement translation; right: proof translation. The tier gap is consistent across all models: controlled, self-contained theorems are 3–10× more tractable than ecosystem-level translations with real library dependencies.

B. Statistical Analysis

B.1. Confidence Intervals

Tables 8 and 9 augment the main-text pass@1 tables with 95% Wilson confidence intervals. The narrow statement-mode intervals (±1–2 pp) reflect the large sample size (n = 4,680); proof-mode intervals are wider (±2–4 pp) due to the smaller evaluation set (n = 888).

Table 8. Statement translation pass@1 with 95% Wilson confidence intervals (n = 4,680 per model).

Model	Passed	Pass@1	95% CI
GPT-5.5	1,360	29.1%	[27.8, 30.4]
DeepSeek-V4-Pro	1,269	27.1%	[25.9, 28.4]
Gemini 3.1 Pro	656	14.0%	[13.1, 15.0]
Qwen3-235B-A22B	452	9.7%	[8.8, 10.5]
Claude Sonnet 4.6	187	4.0%	[3.5, 4.6]

Table 9. Proof translation pass@1 with 95% Wilson confidence intervals (n = 888 per model).

Model	Passed	Pass@1	95% CI
GPT-5.5	93	10.5%	[8.6, 12.7]
Gemini 3.1 Pro	45	5.1%	[3.8, 6.7]
DeepSeek-V4-Pro	13	1.5%	[0.9, 2.5]
Claude Sonnet 4.6	13	1.5%	[0.9, 2.5]
Qwen3-235B-A22B	5	0.6%	[0.2, 1.3]

B.2. Inter-Model Agreement

To assess whether models fail on the same theorems or on complementary subsets, we compute Fleiss' κ across all five models for each (theorem, direction) pair. The kappa values are very low: $\kappa = 0.032$ for statements (n = 4,680) and $\kappa = 0.108$ for proofs (n = 888). Both indicate only slight agreement beyond chance. Models largely fail on different theorems, suggesting that ensembling or best-of-k strategies across models could yield higher pass rates than any single model. The higher proof-mode agreement reflects floor effects: with most models below 2% pass rate, almost all entries are failures.

B.3. Directional Asymmetry

Table 10 reports the full set of directional asymmetries for all six undirected ITP pairs. For each pair (A, B), we report the pooled pass@1 in both directions and the absolute gap. Statement-mode asymmetries are large and systematic: the strongest is Lean 4 ↔ HOL Light ($|\Delta| = 16.4$ pp), driven by HOL Light being an easy target. Proof-mode asymmetries are smaller, consistent with floor effects compressing differences.

Table 10. Directional asymmetry: A ↦ B vs. B ↦ A pass@1 (%), pooled across all models. Pairs are sorted by statement-mode gap.

	Statements			Proofs		
Pair	A ↦ B	B ↦ A	$\lvert\Delta\rvert$	A ↦ B	B ↦ A	$\lvert\Delta\rvert$
L4↔HL	23.0	6.6	16.4	5.7	4.6	1.1
L4↔Isa	17.6	4.3	13.4	1.9	3.2	1.4
Rocq↔HL	23.1	15.4	7.7	4.3	4.6	0.3
L4↔Rocq	15.2	9.4	5.8	3.0	1.4	1.6
Isa↔HL	21.9	25.8	3.9	5.4	3.2	2.2
Rocq↔Isa	19.9	19.0	0.9	2.7	5.7	3.0

The most symmetric statement pair is Rocq ↔ Isabelle ($|\Delta| = 0.9$ pp), despite spanning different logical foundations. This suggests that the translation difficulty between these two systems is roughly balanced: neither is much easier to read from or generate into than the other. In proof mode, the largest asymmetry is Rocq ↔ Isabelle ($|\Delta| = 3.0$ pp), where Isabelle ↦ Rocq (5.7%) outperforms the reverse. Isabelle's verbose Isar proofs may provide richer structural cues for generating Rocq tactics.

B.4. Bradley-Terry Target Ranking

To obtain a principled ranking of ITPs as generation targets, we fit a Bradley-Terry model to pairwise "wins": for each (theorem, model, source) triple, the target that the model successfully translates to "beats" targets that fail. Table 11 reports the log-strength parameters (mean-centered).

HOL Light is the strongest target in both modes. The most striking shift is Isabelle: second-easiest for statements (+0.35) but tied for hardest in proofs (−0.35), consistent with the observation that Isabelle's Isar language is easy to generate as well-formed declarations but difficult for constructing complete proofs. The gap between Lean 4 and the field narrows in proof mode, suggesting that Lean 4's

Table 11. Bradley-Terry log-strength scores for ITPs as generation targets. Higher = easier to generate into. The ranking reverses between modes: Isabelle drops from second-easiest statement target to hardest proof target.

Target ITP	Stmts score	Proofs score
HOL Light	$+0.51$	$+0.55$
Isabelle	$+0.35$	-0.35
Rocq	$+0.15$	$+0.16$
Lean 4	-1.01	-0.35

difficulty as a statement target is partly driven by syntactic strictness rather than proof-search hardness.

B.5. Logical Foundation Pairs

Table 12 breaks down pass rates by the logical foundation of the source and target provers (CIC-based: Lean 4, Rocq; HOL-based: Isabelle, HOL Light).

Table 12. Pass@1 by logical foundation pair. In statement mode, the high CIC $\mapsto$ HOL and HOL $\mapsto$ HOL rates are driven by target-prover effects (Isabelle and HOL Light). In proof mode, all four pairs cluster within 2 percentage points.

		Statements		Proofs	
Pair	Type	Pass/Total	%	Pass/Total	%
CIC $\mapsto$ CIC	within	479/3900	12.3	16/740	2.2
CIC $\mapsto$ HOL	cross	1630/7800	20.9	54/1480	3.6
HOL $\mapsto$ CIC	cross	883/7800	11.3	67/1480	4.5
HOL $\mapsto$ HOL	within	932/3900	23.9	32/740	4.3
Within		1411/7800	18.1	48/1480	3.2
Cross		2513/15600	16.1	121/2960	4.1

The statement-mode asymmetry between CIC $\mapsto$ HOL (20.9%) and HOL $\mapsto$ CIC (11.3%) is not a foundation effect but a target-prover effect: the HOL targets are Isabelle and HOL Light, while the CIC targets are Lean 4 and Rocq (both hard). In proof mode, the four foundation pairs cluster tightly (2.2–4.5%), confirming that logical foundation is not a meaningful predictor of translation difficulty once other factors are controlled.

B.6. Variance Decomposition

Table 13 reports the full logistic regression results from the main-effects model described in Section 4.6. Reference categories are GPT-5.5 (model), statements (mode), Babel-formal (benchmark), and Lean 4 (source and target prover).

An interaction model adding model$\times$mode and model$\times$target interactions improves fit (pseudo-$R^2 = 0.26$), confirming that models have different relative strengths across modes and target provers.

Table 13. Logistic regression coefficients for verification outcome (n = 27,840, pseudo-R^2 = 0.25). Reference categories are GPT-5.5, statements, Babel-formal, Lean 4 source, and Lean 4 target.

Predictor	Coef.	95% CI	p	Note
Intercept	$+2.25$	$[1.6, 2.9]$	$< .001$	
Tgt: HL (vs. L4)	$+2.34$	$[2.2, 2.5]$	$< .001$	Strong tgt
Tgt: Isa (vs. L4)	$+1.96$	$[1.8, 2.1]$	$< .001$	Strong tgt
Mod: DeepSeek (vs. GPT)	—	—	—	
Mod: Qwen (vs. GPT)	—	—	—	
Src: HL (vs. L4)	$+0.24$	$[.09, .4]$	.002	Good src
Same foundation	$+0.19$	$[.07, .3]$	.001	Weak
Src complexity (log len)	-0.42	$[-.5, -.3]$	$< .001$	Harder
Mod: Gemini (vs. GPT)	-0.89	$[-1.0, -.8]$	$< .001$	
Mode: proofs (vs. stmts)	-1.05	$[-1.3, -.8]$	$< .001$	Harder
Bench: F100 (vs. Babel)	-1.50	$[-1.7, -1.3]$	$< .001$	Tier B
Bench: miniF2F (vs. Babel)	-2.33	$[-2.6, -2.1]$	$< .001$	Harder
Mod: Claude (vs. GPT)	-2.39	$[-2.5, -2.2]$	$< .001$	
Src: Rocq (vs. L4)	-0.06	$[-.2, .06]$	.30	n.s.
Src: Isa (vs. L4)	-0.13	$[-.3, .03]$	.11	n.s.
Tgt: Rocq (vs. L4)	-0.15	$[-.4, .05]$	.13	n.s.

B.7. Statements vs. Proofs Correlation

At the theorem level, statement and proof translation difficulty are strongly correlated. Across the 74 theorems where both modes are evaluated (Babel-formal + Formalizing 100 Theorems), the Pearson correlation between per-theorem statement pass rate and proof pass rate is $r = 0.74$ ($p < 0.001$). Crucially, the relationship is asymmetric: all 41 theorems with at least one proof pass also have at least one statement pass, while 32 theorems pass statements but never proofs. No theorem passes proofs but fails all statement translations. This confirms that statement translation success is a near-necessary condition for proof translation success.

C. Dataset and Complexity Characterization

C.1. Dataset Statistics

Table 14 reports corpus-level statistics for the benchmark data. File lengths vary by orders of magnitude across benchmarks and ITPs: miniF2F files are short single-theorem statements (median 202 chars), while Formalizing 100 Theorems proof files range from $\sim$1 KB to $\sim$750 KB, with HOL Light files being the longest due to its verbose OCaml-embedded tactic style.

These proof-file lengths make Formalizing 100 Theorems an end-to-end long-context translation setting: we pass the full source file to the model without chunking or retrieval. Thus, proof translation on this tier reflects both formal translation ability and each model's effective long-context retention; we do not normalize results by context-window size or per-session memory behavior.

Table 15 breaks down statement file lengths by ITP. Isabelle and HOL Light files are consistently longer than their Lean 4

Table 14. Source file length statistics (characters) by benchmark and mode. Per-ITP lengths are averaged across all four provers.

Benchmark	Mode	Files	Mean	Median	Max
Babel-formal	stmts	16	3674	2744	11662
Babel-formal	proofs	16	6240	5118	21262
Formalizing 100 Theorems	stmts	58	16711	6016	289797
Formalizing 100 Theorems	proofs	58	47543	16663	753999
miniF2F	stmts	316	213	202	1499

and Rocq counterparts, reflecting more verbose surface syntax. This length asymmetry is a surface-syntax confound when comparing target difficulty across ITPs.

Table 15. Median statement file length (characters) by ITP and benchmark.

Benchmark	Lean 4	Rocq	Isabelle	HOL Light
Babel-formal	2536	2593	3254	3434
Formalizing 100 Theorems	6467	4733	8525	4566
miniF2F	207	192	270	129

C.2. Source Complexity

Source content length serves as a proxy for proof complexity. In proof mode, longer source proofs are significantly harder to translate (logistic regression coefficient $= -0.54$, $p < 0.001$). In statement mode, the effect is weaker but still negative (coefficient $= -0.10$, $p < 0.001$). Additionally, *failed* proof translations are longer than successful ones (median 3,992 vs. 1,978 characters), suggesting that models over-generate when unable to find a correct proof. This is consistent with the general finding that LLMs struggle with long-context generation tasks and tend to produce verbose, incorrect outputs when the search space is large.

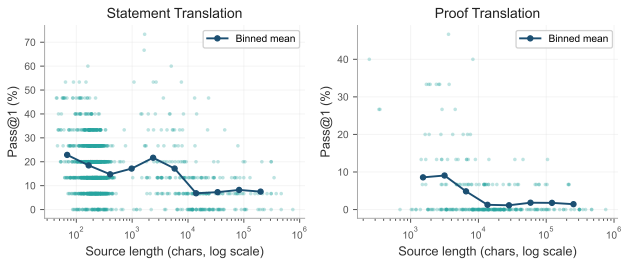

Figure 7. Per-file pass rate vs. source content length (log scale). Left: statement translation; right: proof translation. Dark markers show binned means. The negative trend is stronger for proofs, where each doubling of source length roughly halves the success probability.

C.3. Theorem-Level Difficulty Distributions

Table 16 reports the distribution of per-file pass rates (aggregated across all models, directions, and modes) for each benchmark.

Table 16. Distribution of file-level pass rates by benchmark. Pass rate is computed per file across all models and translation directions. No file exceeds 50% in any benchmark.

Benchmark	Files	Mean	0%	0–10%	10–50%
Babel-formal	16	17.8%	0 (0%)	2 (12%)	14 (88%)
Formalizing 100 Theorems	58	5.0%	1 (2%)	52 (90%)	5 (9%)
miniF2F	316	18.0%	0 (0%)	49 (16%)	267 (84%)

All 16 Babel-formal files are solved at least once, reflecting the controlled tier's tractability. In contrast, only 1 of 58 Formalizing 100 Theorems files has zero passes across all models and directions, but 90% remain below 10% pass rate, representing theorems where ecosystem dependencies make translation very difficult for current models. The miniF2F distribution is more spread: most theorems have nonzero pass rates above 10%, consistent with the statement-only evaluation being more tractable than full proof translation.

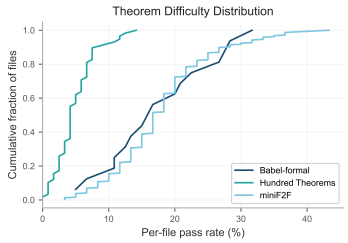

Figure 8. Empirical CDF of per-file pass rates by benchmark. Formalizing 100 Theorems files are concentrated near 0% (90% below 10% pass rate), while Babel-formal and miniF2F files spread across a wider range, reflecting their greater tractability.

C.4. MiniF2F Competition Categories

MiniF2F theorem titles encode their competition source or mathematical topic. Table 17 reports statement translation pass rates by category (all models pooled, statement mode only).

Competition difficulty does not predict translation difficulty. Table 18 groups categories by competition difficulty level. The expected hierarchy AMC < AIME < IMO is *not* reflected in translation pass rates. AIME problems (16.5%) are slightly easier to translate than AMC problems (14.8%), and IMO problems (10.0%) have only a modest gap. These differences should be read as translation statistics for the four-ITP intersection, not as direct measures of mathematical difficulty.

Topic-labeled theorems (algebra, number theory, induction: 15.4% pooled) outperform competition-labeled ones

Table 17. MiniF2F statement translation pass@1 by competition category (all models pooled).

Category	Passed/Total	%	Type
MATH Dataset	2,359/12,120	19.5	Math dataset
Induction	127/720	17.6	Topic
AMC 12B	252/1,440	17.5	Competition
AIME	99/600	16.5	Competition
AMC	103/660	15.6	Competition
Number Theory	54/360	15.0	Topic
Algebra	208/1,440	14.4	Topic
AMC 12A	199/1,560	12.8	Competition
IMO	6/60	10.0	Competition

Table 18. MiniF2F pass rates by competition difficulty level. Mathematical difficulty does not predict cross-ITP translation difficulty.

Difficulty Level	Passed	Total	%
AMC 12 (incl. 12A/B)	554	3660	15.1
AIME	99	600	16.5
IMO / USAMO	6	60	10.0

(15.2%) only marginally in the 4-ITP intersection subset, suggesting that once theorems are formalized across all four systems, topic vs. competition provenance has little impact on translation difficulty.

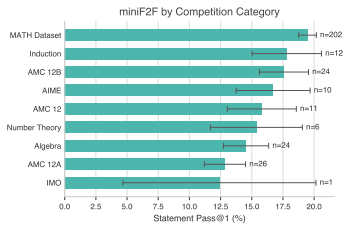

Figure 9. MiniF2F statement translation pass@1 by competition category (all models pooled). Mathematical difficulty level (AMC vs. AIME vs. IMO) does not predict translation difficulty; the MATH Dataset category is easiest to translate despite covering diverse problem types.

D. Extended Related Work

D.1. Formalization tracking and proof exchange

Several projects study overlap among formal libraries without providing a translation benchmark. Formalizing 100 Theorems (Wiedijk, 2008) and the 1000+ Theorems project (1000+ Theorems Contributors, 2026) catalog which mathematical results have been formalized in which proof assistants. These resources are valuable for identifying common mathematical content, but they do not provide aligned source files, translation tasks, or target prover evaluation.

Logical frameworks and proof exchange systems address a complementary problem: representing proofs from different systems in a common foundation. Dedukti (Assaf et al., 2023) is based on the $\lambda\Pi$-calculus modulo theory and has been used as a target for representing several logics and proof systems. Logipedia (Dowek & Thiré, 2023) and EuroProofNet (Thiré, 2020) build on this line of work to support proof interoperability. OpenTheory (Hurd, 2011) provides a package format and standard library for the HOL family of provers. These projects make proof exchange possible in principle, but they do not evaluate LLM generated translations across many directed prover pairs. ITPEVAL is therefore complementary: it treats interoperability as an empirical benchmark problem, with target-prover checking as the evaluation signal.

D.2. Cross ITP translation benchmarks and systems

Babel-formal (Stoskopf et al., 2025) is the closest prior benchmark. It studies Lean and Rocq proof translation using a small aligned corpus of 14 files and 117 lemmas, and explores proof term translation as a bridge between the two systems. ITPEVAL extends the scope from two systems to four, from one logical family comparison to CIC and HOL, and from 14 aligned files to 390 aligned items (1,560 source files across four ITPs).

Recent work on miniF2F in Rocq (Viennot et al., 2025) uses natural language, Lean, and Isabelle sources to produce Rocq theorem statements. This shows that LLMs can help build new formal benchmark ports. However, the task is one target system and statement only. In contrast, ITPEVAL evaluates all 12 source target directions among Lean 4, Rocq, Isabelle, and HOL Light, and includes both statement and proof translation when aligned proofs exist.

Recent prover agents also include cross system translation as one capability. Leanstral (Mistral AI, 2026), for example, is designed around Lean 4 and demonstrates Rocq to Lean translation as part of a broader proof engineering agent. Such systems motivate the need for a benchmark that can measure translation quality systematically rather than through isolated examples.

D.3. Multilingual autoformalization

Multilingual autoformalization studies translation between informal mathematics and multiple formal languages. MMA (Jiang et al., 2023a) constructs informal formal data from Lean and Isabelle and shows that multilingual training can improve autoformalization. ProofWala (Thakur et al., 2025a) studies multilingual proof data synthesis and cross language transfer between Lean and Rocq. KELPS (Zhang et al., 2025) introduces an intermediate representation for translating natural language into Lean, Rocq, and Isabelle.

These works are closely related to our autoformalization and auto-informalization experiment, but differ in direction and evaluation target. They primarily study natural language to formal translation or multilingual training data.

ITPEVAL studies formal-to-formal translation directly and uses natural-language round trips only as a diagnostic for multi-ITP generation context.

D.4. Semantic fidelity and evaluation

A recurring issue in autoformalization is that type checking alone can overestimate correctness. Liu et al. (2025a) introduce BEq (Bidirectional Extended Definitional Equivalence), a neuro-symbolic metric that extends definitional equality with tactic-based transformations to determine equivalence between formal statements; we adopt their bidirectional entailment framework for cross-ITP translation evaluation. Poiroux et al. (Poiroux et al., 2025) study reliable evaluation for statement autoformalization and introduce stronger equivalence based metrics. Liu et al. (Liu et al., 2025b) propose tree edit based evaluation for formal statements, and ASSESS/TransTED (Liu et al., 2026) combines structural and semantic similarity for statement comparison. FormalAlign (Lu et al., 2024) trains models to detect semantic alignment between informal and formal statements, and symbolic equivalence or round trip evaluation methods have also been explored (Li et al., 2024).

Cross-ITP translation adds another layer of difficulty. The source and target statements often live in different kernels, use different libraries, and expose different definitional equalities. This makes direct structural comparison much harder than in single-prover autoformalization. For this reason, ITPEVAL uses target-prover checking as the primary executable signal and BEq as a complementary equivalence-aware check for Lean 4 targets.

D.5. Programmatic prover interfaces

Machine learning for theorem proving has produced several important prover interaction tools. CoqGym (Yang & Deng, 2019) provides a dataset and learning environment for Rocq/Coq proof interaction. LeanDojo (Yang et al., 2023) provides Lean data extraction, retrieval augmented proving, and programmatic interaction with Lean. Pantograph (Aniva et al., 2025) gives a machine to machine interface for Lean 4 and supports proof search. PISA (Jiang et al., 2021) supports interaction with Isabelle. The `itp-interface` project (Thakur et al., 2025b) provides a generic interface for Lean 4 and Rocq.

These systems focus on proof search or data extraction within one prover, or a small subset of provers. ITPEVAL instead requires a verification service that can run four heterogeneous target systems at benchmark scale. The main systems challenge is not tactic search, but reproducible target-prover checking under state isolation. Our unified client, filters, environment-keyed scheduling, checkpointed JSONL verification, and warm isolated backends are designed for this cross-prover evaluation setting.

D.6. Programming language translation benchmarks

Code translation benchmarks such as CodeTransOcean (Yan et al., 2023), HumanEval-X (Zheng et al., 2024), and CRUXEval-X (Xu et al., 2025) evaluate translation between programming languages using tests or executable behavior. They are useful analogues, but ITP translation is different in two ways. First, the target artifact must pass a proof assistant kernel, elaborator, tactic engine, and library environment. Second, translation must preserve logical meaning across systems with different foundations and libraries, not merely produce code with the same input output behavior. These differences motivate a dedicated benchmark for formal translation.

E. Experimental Details

E.1. Autoformalization and Auto-informalization: Full Results

Table 19 reports the full multi-ITP round-trip verification results by model and target prover. Rocq and HOL Light account for most verified outputs, but the pattern is not target-independent: GPT-5.5 is strongest on HOL Light and Rocq, Gemini is strongest on Rocq and relatively strong on Isabelle, and the remaining models show smaller or inconsistent gains. Table 20 compares the multi-ITP Lean 4 results against the single-ITP Lean cycle.

Table 19. Multi-ITP round-trip verification by model and target prover. Entries are pass/checked counts for step 1 and step 3; checked counts exclude empty or skipped outputs.

	Lean 4		Isabelle		Rocq		HOL Light	
Model	S1	S3	S1	S3	S1	S3	S1	S3
GPT-5.5	16/100	17/100	2/100	1/100	61/100	62/100	91/100	92/100
Claude 4.6	12/100	11/100	1/100	1/100	0/100	3/100	11/100	7/100
Gemini 3.1	8/99	9/98	30/97	17/96	61/75	57/67	26/70	28/61
DeepSeek-V4	4/94	7/96	6/94	1/95	14/92	11/94	14/91	18/93
Qwen3-235B	13/98	12/99	3/98	1/99	16/98	13/99	2/99	2/100

Table 20. Lean 4 verification under single-ITP and multi-ITP round trips. Entries count all 100 selected miniF2F theorems per model, treating missing or invalid generations as non-passing.

	Step 1 pass			Step 3 pass		
Model	Single	Multi	Δ	Single	Multi	Δ
GPT-5.5	2	16	+14	4	17	+13
Claude 4.6	13	12	-1	15	11	-4
Gemini 3.1	0	8	+8	0	9	+9
DeepSeek-V4	4	4	0	5	7	+2
Qwen3-235B	5	13	+8	15	12	-3
Pooled	24	53	+29	39	56	+17

Worked example. We trace the full multi-ITP round trip for AIME 1983 Problem 1 (Claude Sonnet 4.6) to illustrate

the process and a characteristic failure mode. The original NL statement is:

Step 0: Reference NL

```
Let x, y and z all exceed 1 and let w
be a positive number such that log_x w
= 24, log_y w = 40 and log_{xyz} w = 12.
Find log_z w.  Show that it is 060.
```

Step 1 (NL $\mapsto$ 4 ITPs) produces four formalizations. Two representative outputs:

Step 1: NL $\mapsto$ Lean 4

```
theorem log_z_w (x y z w :  ℝ) (hx :  x
> 1) (hy :  y > 1) (hz :  z > 1) (hw :
w > 0)
(h1 :  Real.log w / Real.log x = 24)
(h2 :  Real.log w / Real.log y = 40)
(h3 :  Real.log w / Real.log (x * y * z)
= 12) :
Real.log w / Real.log z = 60 := by sorry
```

Step 1: NL $\mapsto$ Isabelle

```
theorem log_z_w:
fixes x y z w ::  real
assumes hx:  "x > 1" and hy:  "y > 1"
and hz:  "z > 1" and hw:  "w > 0"
assumes h1:  "log x w = 24"
assumes h2:  "log y w = 40"
assumes h3:  "log (x * y * z) w = 12"
shows "log z w = 60"
sorry
```

Step 2 (4 ITPs $\mapsto$ NL) produces an informalized description from all four formalizations:

Step 2: 4 ITPs $\mapsto$ NL

```
If x, y, z > 1 and w > 0, and log_x(w) =
24, log_y(w) = 40, and log_{xyz}(w) = 12,
then log_z(w) = 60.
```

Step 3 (NL $\mapsto$ 4 ITPs again) re-formalizes from the step-2 NL. The Lean 4 output:

Step 3: NL $\mapsto$ Lean 4 (second round)

```
theorem log_problem (x y z w :  ℝ) (hx :
x > 1) (hy :  y > 1) (hz :  z > 1) (hw :
w > 0)
(h1 :  Real.log w / Real.log x = 24)
(h2 :  Real.log w / Real.log y = 40)
(h3 :  Real.log w / Real.log (x * y * z)
= 12) :
Real.log w / Real.log z = 60 := by sorry
```

The step-1 and step-3 Lean 4 outputs are syntactically close and express the same real-valued statement (modulo the theorem name), illustrating round-trip stability for this theorem. However, a subtle semantic shift occurs relative to the *reference* formalization: the original miniF2F Lean 4 statement types x, y, z as $\mathbb{N}$ (natural numbers) with $w : \mathbb{N}$, while the round-trip-generated versions use $\mathbb{R}$ (reals), yielding a strictly weaker statement. This information loss is introduced at step 1, where the NL description does not specify integrality, and propagates unchanged through the rest of the round trip. Such type-level semantic drift, invisible in the NL domain, is a characteristic failure mode of NL-mediated round-trips.

E.2. Prompt Templates

All evaluations use a two-message format (system + user) with zero-shot prompting. Placeholders in {braces} are filled per translation pair. The target-language header at the end of each prompt primes generation with the correct output format. System messages instruct the model to return only raw ITP code with no markdown fencing or explanation.

Statement translation

```
Translate the following {src_name}
theorem statement of "{title}" into
{tgt_name}.  Do not prove the theorem;
leave the proof body as sorry or the
equivalent placeholder.

=== {src_name} statement ===
{source_content}

=== {tgt_name} statement ===
```

Proof translation

```
Translate the following {src_name} proof
of "{title}" into {tgt_name}.

=== {src_name} proof ===
{source_content}

=== {tgt_name} translation ===
```

Round-trip: NL $\mapsto$ Lean 4

```
Translate the following natural language
theorem statement into Lean 4.  Do not
prove the theorem; leave the proof body
as sorry.

=== Natural language statement ===
{nl}

=== Lean 4 statement ===
```

Round-trip: Lean 4 $\mapsto$ NL

```
Describe the following Lean 4 theorem
statement in natural language.  Give a
concise mathematical statement of what
the theorem claims.
```

```
=== Lean 4 statement ===
{lean4}

=== Natural language description ===
```

Round-trip: NL ↦ 4 ITPs

```
Translate the following natural language
theorem statement into formal statements
in all four ITPs.  Leave proof bodies as
sorry or the equivalent placeholder in
each system.

=== Natural language statement ===
{nl}

=== Lean 4 ===
=== Isabelle ===
=== Rocq ===
=== HOL Light ===
```

Round-trip: 4 ITPs ↦ NL

```
The following formal theorem statements
all describe the same mathematical
theorem.  Give a concise natural
language description of what the theorem
claims.

=== Lean 4 ===
{lean4}
=== Isabelle ===
{isabelle}
=== Rocq ===
{rocq}
=== HOL Light ===
{hol_light}

=== Natural language description ===
```

E.3. Babel-formal Formalization Process

The 16 Babel-formal files originate in Lean 4 and Rocq (adopted from the Babel-formal benchmark (Stoskopf et al., 2025), plus 2 new files). To complete the four-way alignment required by our benchmark design, we manually formalized all 16 files in Isabelle and HOL Light. This section describes the translation methodology, key design decisions, and illustrative examples.

Design principle: self-contained axiomatization. Every Babel-formal file is designed to be self-contained: it carries its own definitions and axioms rather than depending on any ITP-specific library. This property is inherited from the original Lean/Rocq sources and is preserved in the Isabelle and HOL Light translations. This axiomatized design serves two purposes: (1) it ensures that translation difficulty reflects only foundational differences (type systems, proof languages, abstraction mechanisms) rather than library API mismatches, and (2) it makes the files independently verifiable without installing large library ecosystems.

Illustrative example: `comp_commute`. We show the complete statement-mode file for `comp_commute` (function composition commutativity) in all four ITPs.[3] This file defines composition and identity, then states seven lemmas about commutativity with proof bodies replaced by placeholders, exactly the input an LLM receives during statement translation. The four versions illustrate how the same mathematical content is expressed differently across foundations:

Lean 4

```
universe u v w

namespace CompCommute

variable {a : Type u} {b : Type v} {c : Type w}

def comp {a b c} (g : b -> c) (f : a -> b) : a -> c := fun
↪ x => g (f x)
def id {a} : a -> a := fun x => x

axiom comp_assoc :
  forall {a b c d} (h : c -> d) (g : b -> c) (f : a -> b),
  comp h (comp g f) = comp (comp h g) f
axiom comp_id_l : forall {a b} (f : a -> b), comp (id) f = f
axiom comp_id_r : forall {a b} (f : a -> b), comp f id = f

def commute {a} (f g : a -> a) : Prop := comp f g = comp g f

theorem commute_symm {a} (f g : a -> a) :
  commute f g -> commute g f := by sorry
theorem commute_with_id_l {a} (f : a -> a) : commute f (id)
↪ := by sorry
theorem commute_with_id_r {a} (f : a -> a) : commute (id) f
↪ := by sorry
theorem commute_refl {a} (f : a -> a) : commute f f := by
↪ sorry
theorem commute_congr {a} (f1 f2 g1 g2 : a -> a) :
  f1 = f2 -> g1 = g2 -> commute f1 g1 -> commute f2 g2 :=
↪ by sorry
theorem commute_transport_left_id {a} (f g : a -> a) :
  commute f g -> commute (comp (id) f) g := by sorry
theorem commute_transport_right_id {a} (f g : a -> a) :
  commute f g -> commute f (comp (id) g) := by sorry
end CompCommute
```

Rocq

```
Section CompCommute.
  Universes u v w.
  Definition comp {a b c : Type} (g : b -> c) (f : a -> b)
    : a -> c := fun x => g (f x).
  Definition id {a : Type} : a -> a := fun x => x.

  Class FunProps := {
    comp_assoc : forall {a b c d} (h : c -> d) (g : b -> c)
      (f : a -> b),
      comp h (comp g f) = comp (comp h g) f;
    comp_id_l : forall {a b} (f : a -> b),
      comp (@id b) f = f;
    comp_id_r : forall {a b} (f : a -> b),
      comp f (@id a) = f
  }.

  Definition commute {a : Type} (f g : a -> a) : Prop :=
    comp f g = comp g f.
  Context {FP : FunProps}.

  Lemma commute_symm :
    forall {a} (f g : a -> a), commute f g -> commute g f.
  Proof. Admitted.
  Lemma commute_with_id_l :
    forall {a} (f : a -> a), commute f (@id a).
  Proof. Admitted.
```

[3] The actual source files use Unicode Greek letters (α, β, γ) for type variables in Lean 4 and Rocq, and Unicode symbols ($\Rightarrow$, $\Longrightarrow$, $\equiv$, λ) in Isabelle. We render these as ASCII equivalents for typographic consistency.

```
  Lemma commute_with_id_r :
    forall {a} (f : a -> a), commute (@id a) f.
  Proof. Admitted.
  Lemma commute_refl :
    forall {a} (f : a -> a), commute f f.
  Proof. Admitted.
  Lemma commute_congr :
    forall {a} (f1 f2 g1 g2 : a -> a),
    f1 = f2 -> g1 = g2 ->
    commute f1 g1 -> commute f2 g2.
  Proof. Admitted.
  Lemma commute_transport_left_id :
    forall {a} (f g : a -> a),
    commute f g -> commute (comp (@id a) f) g.
  Proof. Admitted.
  Lemma commute_transport_right_id :
    forall {a} (f g : a -> a),
    commute f g -> commute f (comp (@id a) g).
  Proof. Admitted.
End CompCommute.
```

Isabelle

```
theory comp_commute
  imports Main
begin

definition myComp ::
    "('b => 'c) => ('a => 'b) => ('a => 'c)"
  where "myComp g f == (%x. g (f x))"
definition myId :: "'a => 'a"
  where "myId == (%x. x)"

lemma comp_assoc:
  "myComp h (myComp g f) = myComp (myComp h g) f"
  by sorry
lemma comp_id_l: "myComp myId f = f" by sorry
lemma comp_id_r: "myComp f myId = f" by sorry

definition commute ::
    "('a => 'a) => ('a => 'a) => bool"
  where "commute f g == myComp f g = myComp g f"

lemma commute_symm:
  "commute f g ==> commute g f" by sorry
lemma commute_with_id_l: "commute f myId" by sorry
lemma commute_with_id_r: "commute myId f" by sorry
lemma commute_refl: "commute f f" by sorry
lemma commute_congr:
  "f1 = f2 ==> g1 = g2 ==>
   commute f1 g1 ==> commute f2 g2"
  by sorry
lemma commute_transport_left_id:
  "commute f g ==>
   commute (myComp myId f) g"
  by sorry
lemma commute_transport_right_id:
  "commute f g ==>
   commute f (myComp myId g)"
  by sorry
end
```

HOL Light

```
let myComp = new_definition
  `myComp (g:B->C) (f:A->B) = (\x:A. g (f x))`;;
let myId = new_definition
  `myId = (\x:A. x)`;;

let comp_assoc =
  `!h g f. myComp h (myComp g f)
           = myComp (myComp h g) f`;;
let comp_id_l = `!f:A->B. myComp myId f = f`;;
let comp_id_r = `!f:A->B. myComp f myId = f`;;

let commute = new_definition
  `commute (f:A->A) (g:A->A) <=>
   myComp f g = myComp g f`;;

let commute_symm =
  `!f g. commute f g ==> commute g f`;;
let commute_with_id_l =
  `!f:A->A. commute f myId`;;
let commute_with_id_r =
  `!f:A->A. commute myId f`;;
let commute_refl = `!f:A->A. commute f f`;;
let commute_congr =
  `!f1 f2 g1 g2:A->A.
    f1 = f2 ==> g1 = g2 ==>
    commute f1 g1 ==> commute f2 g2`;;
let commute_transport_left_id =
  `!f g:A->A.
    commute f g ==>
    commute (myComp myId f) g`;;
let commute_transport_right_id =
  `!f g:A->A.
    commute f g ==>
    commute f (myComp myId g)`;;
```

Several differences are visible even in this simple file. Lean 4 uses universe-polymorphic type variables ($\{\alpha$: `Type u`$\}$) with implicit arguments; Rocq bundles the function axioms into a typeclass (`Class FunProps`) and threads the instance via `Context`; Isabelle redefines composition and identity as `myComp`/`myId` (avoiding name clashes with the standard library's `o` and `id`); and HOL Light states lemmas as bare quoted terms (not wrapped in `prove`) since statement-mode files contain no proofs. The placeholder mechanisms also differ: `sorry` (Lean/Isabelle), `Admitted` (Rocq), and omission of `prove` (HOL Light).

For files with algebraic structure hierarchies (e.g., `group`, `ideals`, `lattice_like`), the differences are more pronounced: Lean/Rocq typeclasses become Isabelle locales (e.g., `locale group = fixes mul ... assumes mul_assoc ...`) and HOL Light parameterized predicates (e.g., `is_group (mul:A->A->A) (e:A) (ginv:A->A) <=> ...`). Lean/Rocq class inheritance (e.g., `GroupComm` extending `Group`) becomes locale composition (`locale group_comm = group +`) in Isabelle and predicate conjunction in HOL Light.

Proof translation strategies. Proof styles differ across ITPs. *Isabelle*: We wrote all proofs as structured Isar proofs, using `calc` chains for equational reasoning and Isabelle's automation (`simp`, `metis`, `blast`) for routine steps. *HOL Light*: For simpler files (e.g., `comp_commute`, `group`), we wrote complete tactic proofs. For the `group` file, a key technique is the "babel group" bridge: we embed the parameterized group into HOL Light's native `group` type via a definitional bridge, allowing reuse of the standard `grouptheory.ml` library lemmas while maintaining the axiomatized interface.

For the most complex files (notably `polynomial`), we adopted an *axiomatic* strategy in HOL Light: benchmark lemmas are stated as axioms (`new_axiom`) rather than derived. This is a benchmark engineering choice rather than a proof-exchange claim: (1) the corresponding mathematical development is checked in Lean 4, Rocq, and Isabelle; (2) HOL Light's lack of locale/typeclass abstractions makes certain proof structures (e.g., nested ring–polynomial locale

contexts with 30+ parameters) prohibitively verbose to replicate; and (3) the axioms still serve the benchmark's purpose since the HOL Light file is used only as a *source* or *target* for LLM translation, not as a trust root. Definitions and helper predicates (e.g., `is_poly_context`, which bundles 30+ ring and polynomial axioms into a single predicate) are still fully specified, ensuring that name resolution, type checking, and file structure are exercised by the verifier.

Summary statistics. Across the 16 files (2,902 lines in Lean 4; 2,725 in Rocq), the Isabelle translations total 2,966 lines, nearly isomorphic to the originals, with the slight increase due to locale declaration syntax and explicit `imports` headers. The HOL Light translations total 2,407 lines. Although they are shorter overall, individual lemma signatures are longer because they use explicit universal quantification over structure parameters and the `is_poly_context` bundled-predicate pattern; the axiomatic strategy used for the most complex files reduces proof-side line count. All 16 Isabelle and HOL Light files compile cleanly in Isabelle 2024 and HOL Light (latest git), respectively.

F. Semantic Equivalence Checking Details

This appendix describes the BEq protocol, provides per-model directional breakdowns, documents entailment asymmetry, and gives examples of verified-but-semantically-incorrect translations.

F.1. BEq Protocol

The BEq check tests whether a verified translation preserves the mathematical meaning of the reference theorem. In this paper, BEq is applied to verified Lean 4 target translations. Given a generated Lean statement G and the reference Lean statement R, BEq constructs two entailment problems:

- **Forward** ($G \vdash R$): Assume G as an axiom; attempt to prove R.
- **Backward** ($R \vdash G$): Assume R as an axiom; attempt to prove G.

A translation passes BEq iff both directions succeed. Forward-only means the generated statement is *stronger* than the reference; backward-only means it is *weaker*.

Lean construction. Each directional check is compiled as an independent Lean file:

1. Split the reference file into a header and theorem block; the reference header supplies the Lean imports, `open` commands, and options used for the check.
2. Extract the generated theorem block from the verified model output.
3. Rename the assumed theorem to `stmt_assumed`.
4. Rename the goal theorem to `beq_goal`.
5. Replace the goal proof with a restricted proof search and run Lean.

Deterministic restricted search. Liu et al. (2025a) define BEq through a restricted transformation function: first try Lean's `exact?`, then sample tactic sequences from an LLM under a restricted primitive set. Their ablations show that bidirectionality is important and that restricting the primitive set controls false positives. We preserve this design principle but make the transformation search deterministic. The check first runs `exact?`; the attempt is accepted only if Lean's suggestion explicitly references `stmt_assumed`. If this fails, the checker tries tactics corresponding to the normal primitive set: `exact`, `apply`, `rw`, reverse rewriting, `intro`/`intros`, `constructor`, `ext`, local `have`, case splitting, and existential `use`. Each tactic is instantiated so that a successful proof must use `stmt_assumed`, preventing standalone automation from proving the goal independently of the assumed statement.

Interpretation. BEq is a Lean-checked but incomplete equivalence test. A pass is witnessed by Lean proofs of both entailments. A failure means only that the deterministic restricted search did not establish equivalence; logically equivalent statements can still fail when they require transformations outside the implemented cascade, exceed Lean's search limits, or are not extracted into the expected theorem shape.

F.2. Per-Model Directional Breakdown

Table 21 reports BEq pass rates broken down by model and source prover.

Table 21. BEq pass rate by model and source $\mapsto$ Lean 4 direction (pass/verified).

Model	Rocq $\mapsto$ Lean	Isa $\mapsto$ Lean	HOL $\mapsto$ Lean	Total
Claude Sonnet 4.6	36/43 (83.7%)	1/3 (33.3%)	25/28 (89.3%)	62/74 (83.8%)
DeepSeek-V4-Pro	16/42 (38.1%)	7/18 (38.9%)	14/24 (58.3%)	37/84 (44.0%)
Gemini 3.1 Pro	3/7 (42.9%)	—	—	3/7 (42.9%)
GPT-5.5	5/20 (25.0%)	—	5/9 (55.6%)	10/29 (34.5%)
Qwen3-235B	29/58 (50.0%)	21/40 (52.5%)	12/30 (40.0%)	62/128 (48.4%)
All models	89/170 (52.4%)	29/61 (47.5%)	56/91 (61.5%)	174/322 (54.0%)

BEq pass rates vary by model and source direction. Claude Sonnet 4.6 is strongest overall, while GPT-5.5 and DeepSeek-V4-Pro have low BEq rates despite producing verified Lean statements. Across directions, HOL Light $\mapsto$ Lean has the highest aggregate BEq rate (61.5%), followed by Rocq $\mapsto$ Lean (52.4%) and Isabelle $\mapsto$ Lean (47.5%).

F.3. Forward vs. Backward Entailment

Table 22 reports the breakdown of entailment directions among verified translations.

Table 22. Entailment direction breakdown for verified source ↦ Lean 4 translations. "Both" = BEq pass; "BwdOnly" = reference entails generated but not vice versa (weaker translation).

Model	Checked	Both	Neither	FwdOnly	BwdOnly
Claude Sonnet 4.6	74	62	10	0	2
DeepSeek-V4-Pro	84	37	40	0	7
Gemini 3.1 Pro	7	3	4	0	0
GPT-5.5	29	10	18	0	1
Qwen3-235B	128	62	60	0	6
All models	322	174	132	0	16

All 16 asymmetric cases are backward-only; no model produces any forward-only cases (translations stronger than the reference). This pattern is consistent with *weakening*: dropping hypotheses, loosening types, or shifting to a more general formulation that the reference can still imply. Most failures, however, fail in both directions (132 cases). Additionally, 26 verified records have extraction errors and are counted as BEq failures in the aggregate results.

F.4. Failure Examples

We illustrate three classes of BEq failures from DeepSeek-V4-Pro's backward-only cases.

Genuine semantic weakening. Theorem `amc12a_2021_p3` asks to show $x - y = 14238$ given constraints on x, y. The reference uses integer coercion (`↑x - ↑y = (14238:ℤ)`), while DeepSeek generates natural-number subtraction (`x - y = 14238`). Since $\mathbb{N}$ subtraction truncates at zero, the generated statement is strictly weaker. The reference entails the generated (backward), but not vice versa (forward).

Definitional equivalence beyond the cascade. Theorem `amc12a_2003_p5` encodes digit constraints. The reference uses `Nat.ofDigits 10 [0,1,C,M,A]`, while DeepSeek inlines the arithmetic as `10000*A + 1000*M + 100*C + 10*1 + 0`. These are definitionally equal in Lean 4, but the tactic cascade cannot close the forward direction. This is a false negative: the statements are logically equivalent, but BEq reports backward-only.

Structural reformulation. Theorem `amc12_2000_p1` has the reference state hypotheses as separate binder arguments (`(h₀ : i ≠ m ∧ ...) (h₁ : i*m*o = 2001)`), while DeepSeek bundles everything into a single implication (`(i ≠ m ∧ ... ∧ i*m*o = 2001) → ...`). The two formulations are logically equivalent, but the cascade hits a recursion depth limit on the forward direction. This is another false negative from tactic limitations.

Limitations. The tactic cascade is deliberately conservative: it uses only standard Lean 4 automation and does not attempt custom proof search. This means that some backward-only and neither-direction cases may be logically equivalent to the reference but beyond the cascade's reach. Prototype extensions to Rocq (using `auto`, `eauto`, `lia`, etc.), Isabelle (using `auto`, `blast`, `sledgehammer`, etc.), and HOL Light (using `MESON_TAC`, `ARITH_TAC`, etc.) exist, but this paper reports only the Lean 4 deployment; evaluating non-Lean BEq at scale remains future work.

G. Infrastructure Details

Section 3.3 rests on one invariant: every accelerated backend must be observationally equivalent to checking the generated artifact in isolation. Concatenating generated files into one prover session is unsound for this benchmark because later artifacts could depend on earlier declarations and top-level-name collisions could turn otherwise valid artifacts into failures.

Toolchain manager. The released `itpeval` CLI installs and runs proof assistants under a common prefix. Each prover adapter exposes installation and checking entry points, and `run_itp()` materializes a prover-specific task directory before invoking the native binary. The adapter returns a structured result containing exit status, wall-clock time, captured logs, and backend metadata. Adding a new prover requires only an adapter that maps generated source text to a native check task and returns this result object.

State-isolated resident verification. Cold checking is simple but too slow for large cross-ITP evaluation. The resident layer therefore amortizes startup while preserving per-file semantics. Lean requests are fresh synthetic files, Rocq requests are followed by `Reset Initial`, Isabelle requests are temporary theories in server sessions, and HOL Light requests use either collision-aware worker packing or a forkserver whose child exits after each request.

Table 23. Resident backend semantics. Each backend amortizes setup cost while preserving one generated artifact as the evaluation unit.

Prover	Backend	Reuse	Isolation
Lean 4	pool/server	compiler workers	fresh file
Rocq	REPL pool	persistent proc.	`Reset Initial.`
Isabelle	server-batch	persistent session	temp. theories
HL	resident pool	preloaded OCaml	collision packing
HL	forkserver	preloaded parent	child exit

Scheduling, caching, and bisection. Each generated record is assigned an environment key consisting of the

target prover, benchmark mode, imports, and backend constraints. The scheduler groups compatible records, orders groups by estimated cost, and dispatches them to the fastest backend that preserves per-file semantics. Strict filters are deliberately one-sided: they can reject invalid artifacts, such as proof-mode outputs containing placeholders, but they can never turn an artifact into a success. For batch-capable backends, failures are localized by adaptive bisection until per-record labels are recovered.

Checkpointing and telemetry. Verification writes append-only JSONL checkpoints keyed by stable record identifiers, so interrupted runs resume without losing completed labels. Each checked record stores the original generated text, pass/fail label, first diagnostic, wall-clock time, backend name, environment key, scheduling metadata, and cache status. This telemetry distinguishes model failures from verifier or toolchain failures and enables cold-equivalence audits of resident backends.

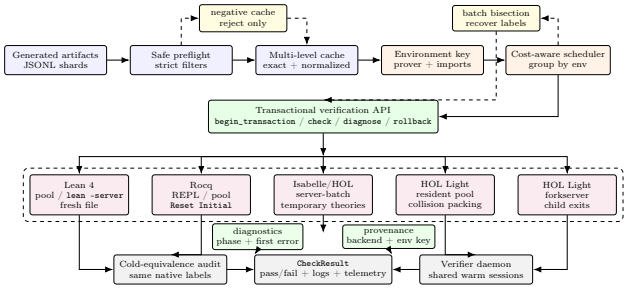

Figure 10. Expanded infrastructure view for the state-isolated verification service. The same architecture is summarized in the main text.

G.1. Reproducibility

All code, data, and evaluation artifacts are available in the public `itpeval` release. The repository includes: (1) the complete benchmark data in structured JSON format (1,560 statement files and 296 proof files across four ITPs), (2) the evaluation pipeline with both batch and synchronous execution modes supporting all five models, (3) bootstrap scripts that install all four ITP toolchains on macOS and Linux, (4) all raw model outputs and native ITP verification results, and (5) scripts to reproduce all tables and figures in this paper.

The evaluation pipeline is designed for end-to-end reproducibility. LLM translation and ITP verification are decoupled: translations can be generated on any machine with API keys, while verification requires the ITP toolchains and can run separately on the same pre-generated outputs. All runs produce checkpointed JSONL files that support safe interruption and resumption. The pipeline is also extensible: new models can be added with a single configuration line, new providers by implementing a dispatch function, and new benchmark data by appending records to the data files (every theorem must have all four ITP variants).

Compute resources. All LLM generation was performed through provider APIs (OpenAI, Anthropic, Google, OpenRouter); no local GPU compute was required. ITP verification is CPU-only and runs on any machine with the four toolchains installed. Per-record verification telemetry (wall-clock time, backend provenance, cache status) is included in the released JSONL checkpoints.

The autoformalization round-trip experiment uses a curated 100-record subset of miniF2F, selected to avoid the test split's $\sim$62% `mathd` dominance (all 60 non-`mathd` test records plus 40 randomly sampled `mathd` records, seed=42). Both single-ITP (NL $\mapsto$ Lean 4 $\mapsto$ NL $\mapsto$ Lean 4) and multi-ITP (NL $\mapsto$ 4 ITPs $\mapsto$ NL $\mapsto$ 4 ITPs) cycles are included.