When Does Collaboration Help in Lean Proof Search?

Coverage gains without a measurable benefit from partial-proof exchange

Jacob Belenkii

Abstract

When two language models work on the same Lean theorem, they can help in two different ways. They can provide **coverage**, by trying different searches and succeeding on different problems, or **transfer**, by sending information that makes the other search more likely to succeed. I built a compiler-guided proof-search harness around Qwen 3.5 Flash and GPT-OSS 120B and evaluated both mechanisms.

Each model repeatedly proposed and repaired a Lean proof using compiler feedback. I compared either model alone, the two models searching in parallel without communication, and the same parallel search with verified partial proofs exchanged between them. On four supplied problems with mixed preliminary outcomes, the silent two-model system solved 19 of 24 runs, compared with 16 for Qwen alone and 9 for GPT-OSS alone. The communicating system solved 17. On a replication set deliberately selected to be difficult or variable for Qwen, the models had complementary outcomes, but message passing again did not improve the result: the silent and communicating mixed-model systems solved 23/32 and 22/32 runs. A direct Qwen+Qwen control solved 29/32.

The same-model result changes the interpretation. Qwen+Qwen completed 671 model calls, versus 489 for silent Qwen+GPT-OSS, because Qwen was much faster. Under a fixed wall-clock budget, model choice changed both the diversity of the searches and how much search was completed. Two recorded-state replays also found no improvement in the receiver's next response: 151/376 versus 152/376 warm-compiling responses on 47 earlier-study states, and 6/208 versus 8/208 on 26 replication-set states, with and without the message respectively. These experiments show useful coverage from repeated search, including some unique GPT-OSS successes, but no measured transfer benefit from the partial-proof protocol tested here.

1. The question: what kind of collaboration can help?

The take-home asks whether a two-model system outperforms either model alone and, when it does, what one model contributes that the other lacks. A useful way to answer that question is to separate two effects that are often both called collaboration.

Coverage is the benefit of having another chance to solve the problem. The searches need not communicate. If their failures do not perfectly overlap, accepting the first verified proof can outperform either search alone.

Transfer is an additional benefit caused by communication. One model sends a partial result, plan, or criticism, and receiving it changes what the other model can solve. Transfer is present only if a communicating system outperforms the same system with communication removed.

This distinction also provides a design lens for the harness. Each problem has a dollar cap and a wall-clock cap. The harness must decide how to spend that budget: repair the current candidate, restart from a different approach, run another search in parallel, or use a peer's partial proof. I focused primarily on allocating model calls. A call is not a uniform unit of compute - GPT-OSS used more reasoning tokens and was substantially slower - so I report realized calls and time and do not claim token- or dollar-optimality.

The main result is straightforward. Multiple searches provided coverage. Qwen and GPT-OSS sometimes solved different runs, especially on problems selected to be difficult for Qwen. However, the partial-proof messages tested here did not add measurable transfer. A late same-model control also showed that the best allocation on the replication set was two Qwen searches, not one search from each model, despite GPT-OSS's unique successes.

2. Related work and positioning

This study sits at the intersection of verifier-guided proof search and algorithm portfolios. Systems such as COPRA execute proposed tactics inside Lean or Coq and feed the resulting state back into a stateful search [3]. LeanDojo similarly treats interaction with Lean, premise retrieval, and search as first-class parts of the proving system [4]. Baldur instead generates a whole proof and then asks a model to repair it using verifier errors [5]. These systems support the basic engineering premise of this project: a model sample is one move in a search process, not a complete method by itself.

The compiler-guided repair mechanism was inspired most directly by APOLLO [6]. APOLLO analyzes failed proofs, isolates failing sublemmas, applies automated solvers, and asks a model to finish remaining goals before reassembling the proof. The final harness here is substantially simpler and is not an implementation of APOLLO: it runs repeated whole-file repair in two peer tracks and, when communication is enabled, sends only a warm-compiling prefix with explicit holes. The resemblance is therefore at the level of using Lean feedback to recover structure from failed generations.

A particularly close contemporary comparison is LEAP [10], which places a general-purpose frontier model inside a prover with informal planning, compiler-guided revision, verified proof sketches, an AND-OR dependency graph, and LLM review. Both projects ask whether scaffolding can make general-purpose models effective Lean provers. But LEAP evaluates a structured end-to-end prover, reported with Gemini 3.1 Pro as the backend; it neither compares two model families as peer searchers nor isolates communication against a silent control. Its proof sketches also preserve a stronger object than these prefix packets: they verify that a parent goal follows from named child lemmas. This report instead studies low-budget allocation between two parallel searches and tests whether a compiling prefix changes the receiver's success.

The allocation question has an older analogue in algorithm portfolios: when randomized solvers have different performance profiles across instances and seeds, a fixed budget can be divided among them rather than committed to one run [1]. Repeated same-model sampling, commonly summarized by pass@k, is the corresponding homogeneous portfolio [2]. Specialized Lean models such as DeepSeek-Prover-V2 and Goedel-Prover show how model specialization can change the available search policies [7, 8]. Their benchmark scores alone do not establish paired error complementarity with a general-purpose model; that requires observing whether one system rescues the other's failures. This report makes that paired comparison directly and separately tests whether communication adds anything beyond parallel search.

3. System design

3.1 Compiler-guided repair

One solver run starts with the theorem statement and may make up to 25 model calls. After every proposed proof, a warm Lean process checks the candidate and returns bounded compiler diagnostics. The next prompt contains the current candidate and those diagnostics, so the model can repair the proof it actually wrote. If the same candidate or error pattern repeats, the harness restarts from the original theorem while retaining a short memory of abandoned approaches. A small deterministic tactic cascade is tried before model calls.

When a failed proof has a useful prefix, the harness attempts to preserve it. It replaces the unresolved part with explicit `sorry` holes and checks the result in Lean. If the resulting skeleton compiles, the harness has a verified partial proof: the preserved declarations and proof steps type-check, while the remaining work is explicit.

A warm Lean pass is provisional. The returned file counts as a success only if a fresh run of the pinned grader-compatible Comparator accepts it. The system returns the first candidate that passes that check.

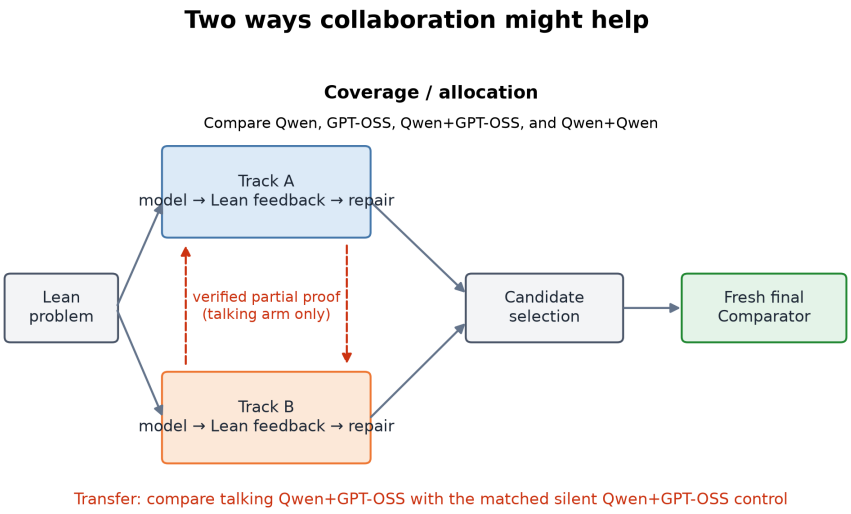

Figure 1. *Solo and portfolio policies measure coverage and search allocation; talking versus silent Qwen+GPT-OSS isolates transfer through the tested partial-proof message.*

3.2 Five allocation policies

The experiments compare five policies:

Policy	Search allocation	Communication
Qwen solo	One Qwen repair search	None
GPT-OSS solo	One GPT-OSS repair search	None
Silent Qwen+GPT-OSS	One search from each model	None
Talking Qwen+GPT-OSS	One search from each model	Verified partial proofs
Silent Qwen+Qwen	Two independent Qwen searches	None

Table 1. *Allocation policies evaluated in the study.*

The two mixed-model policies use the same models, limits, repair logic, and scheduling rule. Because Qwen often exhausted its calls before slower GPT-OSS produced a partial proof, both policies reserve one final Qwen call. In the talking policy, a message can release that call; in the silent control, it is released after GPT-OSS finishes or shortly before the dispatch deadline. This makes talking versus silent the cleanest test of transfer.

The message contains the peer's warm-compiling skeleton, its remaining Lean goals, and any completed helper lemmas. The receiver is asked to preserve what compiles and fill the holes. Newer messages replace older ones. Earlier development versions sent a whole failed proof and diagnostics; those pilots motivated the final format but are not pooled into the primary comparison.

The Qwen+Qwen policy is a direct test of a different allocation: spend both parallel searches on the faster model. It uses two separately identified Qwen tracks with distinct provider sub-seeds and no reserved call. Because this policy differs in both model composition and scheduling, it estimates the performance of the complete policy rather than the isolated causal effect of changing GPT-OSS to Qwen.

4. Evaluation

4.1 Problems and repeated runs

I used two evaluation sets.

The first comes from the 16 supplied problems. Preliminary runs separated problems that were almost always solved, almost never solved, or had mixed outcomes. To put most of the limited budget where arm differences could be observed, the primary supplied-set comparison repeated the four mixed-outcome problems six times, for 24 problem-by-repetition groups. The selection rule was fixed before this comparison. Because the main table is conditional on that four-problem selection, Appendix A, Table A1 reports scores for all 16 supplied problems.

The second set tests whether the conclusions transfer beyond the supplied problems. I took the first 32 miniF2F test statements, in benchmark order, that compiled with the pinned Mathlib import and were not closed by the deterministic tactic cascade. Eight short Qwen runs per problem identified problems with mixed Qwen outcomes. I selected the eight closest to four successes in eight runs and evaluated every primary policy four times, producing 32 groups per policy.

Qualitatively, this replication set is narrower than the supplied set: it contains four AIME algebra problems, one linear system, two real inequalities, and one existential irrational-power result. The supplied set also includes modular arithmetic, divisibility and extremal problems, sequences, combinatorial sums, and Diophantine number theory, as well as multi-theorem and numeric-answer interfaces. The supplied `problem.md` files give natural-language statements; the replication descriptions only identify the miniF2F source and refer the model to the formal Lean statement. The replication therefore adds an external benchmark source, but not representative topic or prompt-format coverage.

This replication set is intentionally conditional on Qwen. It is useful for studying allocation where Qwen has room to improve, but it is not a random miniF2F sample. Its pass rates and the frequency of GPT-OSS wins must not be generalized to the full benchmark.

4.2 Outcomes and comparisons

Every launched run is scored by the final Comparator result. The main tables report per-problem successes as well as totals. Conditions for the same problem and repetition index were scheduled together, but OpenRouter sampling was not deterministic; the index is a matching label, not a guarantee that two arms share the same random draw.

Coverage is measured by the solo, mixed-model, and Qwen+Qwen pass rates. Transfer is measured by talking minus silent within the matched mixed-model design.

There is also a useful benchmark for allocating the second search. Under statistical independence, two searches form the same probability model as a two-component parallel system: the combined attempt fails only if both components fail [9]. If a Qwen search and a GPT-OSS search have per-problem success probabilities p_Q and p_G, their chance of at least one success is

$$1 - (1 - p_Q)(1 - p_G).$$

A second independent Qwen search has success probability $1 - (1 - p_Q)^2$, the two-sample special case of pass@k [2]. Under that assumption, the mixed-model advantage over two Qwen searches is

$$\begin{aligned}[1 - (1 - p_Q)(1 - p_G)] - [1 - (1 - p_Q)^2] \\ = (p_G - p_Q)(1 - p_Q).\end{aligned}$$

This gives a simple tipping point, not a fitted law: under independence and equal resources, the mixed allocation wins when $p_G > p_Q$. More generally, a different second model is worth using when it rescues failures of the first Qwen search more often than another Qwen search would. Let Q_1 denote the first Qwen search, Q_2 a fresh Qwen search with a different seed, and G a GPT-OSS search on the same problem. Without assuming independence, the mixed-model allocation is preferable when

$$\Pr(G \text{ succeeds} \mid Q_1 \text{ fails}) > \Pr(Q_2 \text{ succeeds} \mid Q_1 \text{ fails}).$$

Here "fresh search" means that Q_2 runs without communication or shared search state; it does not assume that Q_1's and Q_2's outcomes are statistically independent. The comparison also assumes that the candidate second searches receive comparable resources. In the actual harness, their realized calls and latency differed, so the direct system comparison reflects both conditional rescue and search throughput.

I use the formula only as an independence benchmark. The direct Qwen+Qwen arm is more informative because it includes the actual scheduler, latency, and stopping behavior. Estimates based on four solo runs per problem are necessarily noisy.

5. When did another search provide useful coverage?

5.1 Supplied problems: Qwen was the better use of a call

On the four supplied problems with mixed preliminary outcomes, Qwen solved 16/24 runs and GPT-OSS solved 9/24. The silent mixed-model policy solved 19/24.

Problem	Qwen	GPT-OSS	Q+G silent	Q+G talking
p03: square inequality	6/6	6/6	6/6	6/6
p07: least divisible	4/6	1/6	6/6	4/6
p08: sum of products	6/6	2/6	6/6	6/6
p09: IMO 1964	0/6	0/6	1/6	1/6
Total	**16/24**	**9/24**	**19/24**	**17/24**

Table 1. *Per-problem results on the four supplied mixed-outcome problems. Each cell reports successful runs out of six.*

The cumulative search curves tell the same story at the level of observed calls. At matched positions in a solver run, Qwen's cumulative solve rate was never below GPT-OSS's on the supplied set. GPT-OSS produced only one solo success in a problem-and-repetition group where Qwen failed. The silent portfolio improved on one Qwen run, but these data do not show that GPT-OSS was a better second allocation than another Qwen search; a direct Qwen+Qwen control was not run on this set.

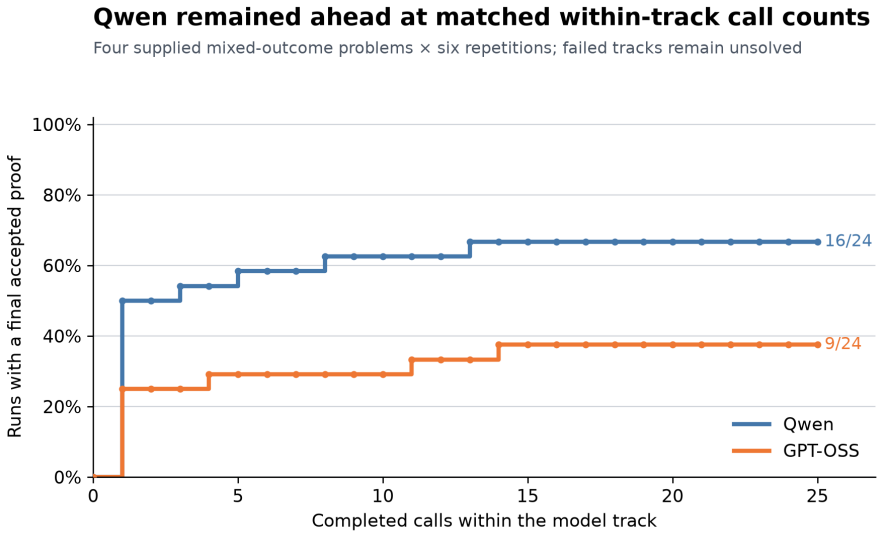

Figure 2. *Cumulative final-Comparator success by the call index of the first accepted candidate. The 24 runs per model cover four supplied mixed-outcome problems at six repetitions. The horizontal axis is completed calls within a track, not elapsed time or equal total system cost; failed tracks remain unsolved.*

The repair loop itself also mattered. On `p10_factorial_pow`, the supplied baseline agent failed both observed runs, while the compiler-guided engine succeeded in 14/14 runs across two later matched waves. This is not a perfectly controlled ablation - the reasoning settings and surrounding engine also changed - but it shows why repeated search should be made reliable before adding a communication protocol.

5.2 Qwen-hard replication: complementary failures, but Qwen+Qwen won

The replication set was different. Across 32 solo runs, Qwen succeeded 18 times and GPT-OSS 14 times. Their outcomes were not nested: both succeeded in 7 groups, only Qwen succeeded in 11, only GPT-OSS succeeded in 7, and neither succeeded in 7. GPT-OSS therefore contributed real coverage on problems selected to be difficult or variable for Qwen.

Problem	Qwen	GPT-OSS	Q+G silent	Q+G talking	Q+Q silent
AIME 1983 P1	2/4	1/4	2/4	0/4	4/4
AIME 1990 P15	3/4	4/4	4/4	4/4	4/4
AIME 1990 P4	1/4	1/4	2/4	3/4	4/4
AIME 1997 P9	4/4	0/4	2/4	3/4	4/4
Two-variable linear equations	4/4	2/4	4/4	3/4	3/4
9/(x+y+z) inequality	1/4	0/4	2/4	1/4	4/4
AM-GM sum-of-squares	3/4	3/4	4/4	4/4	3/4
Irrational-power construction	0/4	3/4	3/4	4/4	3/4
Total	**18/32**	**14/32**	**23/32**	**22/32**	**29/32**

Table 2. *Per-problem results on the Qwen-hard/variable replication set. Each cell reports successful runs out of four.*

The silent mixed-model portfolio gained five successes over Qwen alone. However, the direct Qwen+Qwen policy gained eleven and finished six runs ahead of the silent mixed policy. In the paired comparison, Qwen+Qwen alone succeeded in nine groups, the mixed policy alone succeeded in three, both succeeded in twenty, and neither failed both. With only 32 selected groups, this is not a benchmark-wide model ranking. It is a clear policy result on the evaluated set.

Realized search helps explain why the marginal solo rates were not enough. The nominal ceiling was 25 calls per track, but most tracks stopped earlier because another track succeeded or the dispatch window closed. Qwen+Qwen completed 671 calls across its 32 runs. Silent Qwen+GPT-OSS completed 489: 319 Qwen calls and 170 GPT-OSS calls. Two fast Qwen tracks therefore bought substantially more search under the same outer wall-clock environment.

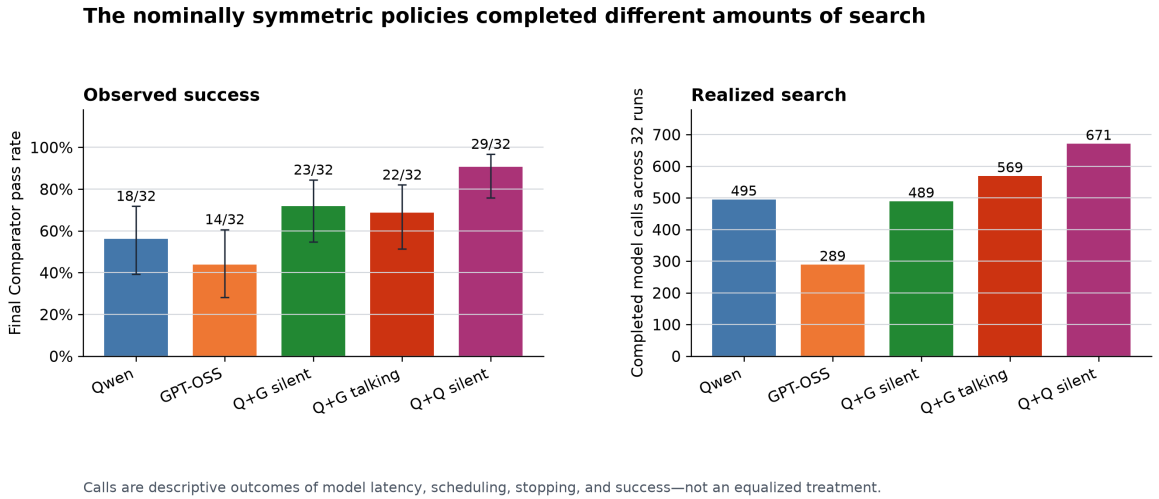

Figure 3. *Final pass rates and realized model calls on the selected replication set. Error bars are Wilson 95% intervals. Completed calls reflect latency, scheduling, early stopping, and success; they are not an equalized treatment or a causal dose-response.*

This does not prove that throughput caused the six-run difference. The arms also used different provider sub-seeds, and the mixed policy had the reserved-call rule needed for the communication experiment. It does establish the broader design point: deciding whether to diversify across models requires measuring actual rescue rates and actual search completed, not only marginal solo pass rates.

6. Did partial-proof exchange add transfer?

6.1 Talking versus silent search

The talking and silent mixed-model policies isolate the tested communication mechanism. On the supplied mixed-outcome problems, talking solved 17/24 and silent solved 19/24. On the Qwen-hard replication, talking solved 22/32 and silent solved 23/32.

In both studies the point estimate is slightly negative. The samples are too small to rule out modest positive or negative effects, but they provide no evidence that the verified partial-proof messages increased final success. This conclusion is intentionally limited to the models, prompts, scheduling rule, and packet-style communication tested here.

The logs also prevent a common attribution error. A talking run can beat its silent counterpart by chance even if no message affected the successful proof. For each run I recorded whether a partial proof was produced, delivered before the receiver stopped, included in a completed request, reflected in the next candidate, accepted by the warm Lean check, and used in the final proof. No reported success is attributed to communication without this audit trail.

6.2 Counterfactual replay at recorded prompt states

End-to-end pass rates are a blunt instrument: each theorem yields one binary result, and useful messages are produced only in some runs. I therefore used recorded-state replay to test the message at exactly the point where the receiving model saw it.

The first replay covered every packet-exposed request in the main earlier-study archives: 47 source prompt states. For each state, I sampled the same receiving model eight times with the recorded message and eight times after deleting only the message block and its instruction. Every next response was checked in the same warm Lean environment. The response compiled in 151 of 376 samples with the message and 152 of 376 without it.

Two additional message variants gave the receiver only the helper lemmas or only the first three lines of the compiling prefix. They produced 149/376 and 141/376 compiling responses, respectively, against the same 152/376 no-message baseline. None showed an immediate improvement.

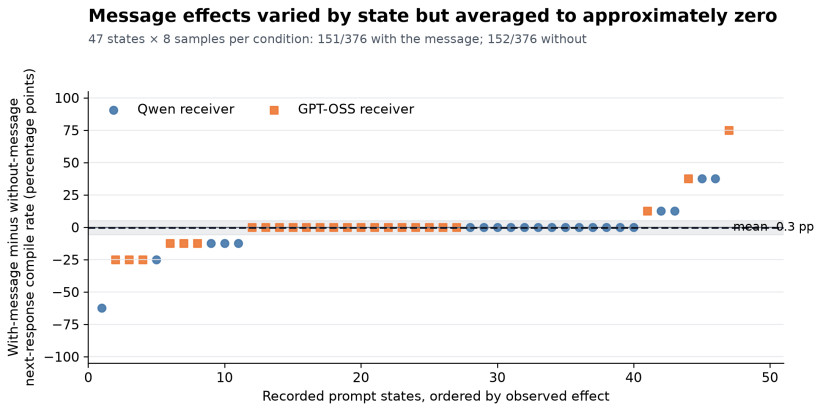

Figure 4. *Earlier-study replay. Each point is one recorded prompt state and compares eight samples with and without the message. The shaded band is a descriptive 95% bootstrap interval across 47 source states; the responses are not independent theorem instances.*

I repeated the same procedure on all 26 packet-exposed states from the Qwen-hard replication study. The next response compiled in 6/208 samples with the message and 8/208 without it, a difference of -1.0 percentage point. The baseline compile rate was much lower in this second population, so I report the two replays separately rather than pooling their responses.

Replay population	Source states	With message	Without message	Difference
Earlier-study replay	47	151/376	152/376	-0.3 pp
Replication-set replay	26	6/208	8/208	-1.0 pp

Table 3. *Recorded-state replay results. The populations are reported separately because their baseline next-response compile rates differ.*

The replays answer a specific causal question: at these recorded states, did adding this message change the probability that the receiver's next response compiled? They do not measure effects on later repair steps or prove that all partial-proof exchange is ineffective. Each replay repeatedly samples a small set of recorded states; its responses are not independent theorem instances.

7. What the traces suggest

The transcripts show that the models noticed and sometimes copied the messages. In one `p06_pow_mod` state, a Qwen skeleton already contained the two key arithmetic facts and left one hole that `norm_num` could close. GPT-OSS often completed that nearly finished proof when given the packet. When the same packet was truncated to three lines, the effect weakened.

The opposite pattern appeared on `p07_least_divisible`. A packet carried a wrong decomposition, and the receiver often preserved it as instructed. In those recorded states, the message reduced rather than increased the next-response compile rate.

These cases motivate a limited mechanism hypothesis. Salvage guarantees that the transmitted prefix compiles, but it also removes the first unresolved portion of the proof. The exchanged message therefore often contained the part of the proof that was already working, but not the unresolved step that had prevented the sender from completing the proof. A nearly complete proof was useful because little inference remained; a wrong partial plan could anchor the receiver.

LEAP's verified-sketch design [10] illustrates a more structured alternative. Rather than cutting a proof at its first failure, it asks the model to name supporting lemmas and verifies that proving those lemmas would complete the parent goal. That representation keeps the missing bridge explicit and makes progress reusable across branches. I did not implement or evaluate that mechanism here, so it is a concrete follow-up design rather than an explanation of the present results.

That account is consistent with the replay, but it is not proved by it. A compiling prefix can contain the main mathematical insight, and a failed suffix can be mere formal plumbing. Other communication designs remain open: assigning a peer a named subgoal, asking for an independent critique rather than preservation, or exchanging several alternative sketches instead of one latest prefix.

8. Answer and limitations

These experiments suggest three practical answers to "when does collaboration help?"

First, another search helps when it has a substantial chance of rescuing failures from the first search. That rescue can come from a different model or another sample from the same model. Average model accuracy is not enough; the relevant comparison is the success of each complete allocation policy under the actual budget.

Second, communication helps only when the information changes the receiver's trajectory. Delivery is necessary but not sufficient. Here the messages reached both receivers, yet neither the matched end-to-end comparisons nor the recorded-state replay showed a benefit from the tested verified-prefix protocol.

Third, system throughput is part of the scientific answer. The slower model completed fewer calls, so a nominally symmetric two-track design did not provide equal search. On the Qwen-hard replication set, GPT-OSS had unique successes, but two Qwen tracks completed more calls and achieved the highest observed pass rate.

The scope is narrow. The study uses two fixed models, Lean 4 competition problems, one repair-loop family, and four closely related partial-proof formats. The replication problems were selected using Qwen and are not representative of miniF2F as a whole. Provider

sampling was nondeterministic, the number of repeated problems was small, and calls were not matched for tokens, dollars, or latency. The results therefore support a design recommendation, not a universal law: before building elaborate interaction, compare against both a silent mixed-model portfolio and a same-model repeated-search control, measure the calls that actually finish, and use targeted replay to test whether messages change the receiver at all.

References

[1] C. P. Gomes and B. Selman. "Algorithm portfolios." Artificial Intelligence 126(1-2):43-62, 2001. doi:10.1016/S0004-3702(00)00081-3.

[2] M. Chen et al. "Evaluating Large Language Models Trained on Code." arXiv:2107.03374, 2021.

[3] A. Thakur, G. Tsoukalas, Y. Wen, J. Xin, and S. Chaudhuri. "An In-Context Learning Agent for Formal Theorem-Proving." COLM, 2024. arXiv:2310.04353.

[4] K. Yang et al. "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models." NeurIPS, 2023. arXiv:2306.15626.

[5] E. First, M. N. Rabe, T. Ringer, and Y. Brun. "Baldur: Whole-Proof Generation and Repair with Large Language Models." ESEC/FSE, 2023. doi:10.1145/3611643.3616243.

[6] A. Ospanov, F. Farnia, and R. Yousefzadeh. "APOLLO: Automated LLM and Lean Collaboration for Advanced Formal Reasoning." NeurIPS, 2025. arXiv:2505.05758.

[7] Z. Z. Ren et al. "DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition." arXiv:2504.21801, 2025.

[8] Y. Lin et al. "Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving." arXiv:2502.07640, 2025.

[9] NIST/SEMATECH. "Parallel or redundant model." e-Handbook of Statistical Methods, sec. 8.1.8.3.

[10] P.-N. Kung et al. "LEAP: Supercharging LLMs for Formal Mathematics with Agentic Frameworks." arXiv:2606.03303v2, 2026.

Appendix A. Complete supplied-set and paired results

Table A1 gives results for every supplied problem. Its columns come from two descriptive screens: the solo columns use one fixed-seed run per model and problem, while the mixed-model preliminary columns use three repetitions on six core problems and one on the other ten. They are not pooled with the six-repetition primary comparison in Table 1.

Problem	Qwen solo	GPT-OSS solo	Q+G silent prelim.	Q+G talking prelim.
p01: linear equation	1/1	1/1	1/1	1/1
p02: fraction cancellation	1/1	1/1	1/1	1/1
p03: square inequality	1/1	1/1	3/3	3/3
p04: sum of squares	1/1	1/1	1/1	1/1
p05: GCD / Mersenne	1/1	1/1	1/1	1/1
p06: power modulo	1/1	1/1	3/3	3/3
p07: least divisible	1/1	0/1	3/3	2/3
p08: sum of products	1/1	0/1	1/1	1/1
p09: IMO 1964	0/1	0/1	1/1	0/1
p10: factorial power	1/1	0/1	2/3	3/3
Putnam 2018 A1	0/1	0/1	0/1	0/1
Putnam 2020 A2	0/1	0/1	0/3	0/3
RMO 2000/2	0/1	0/1	0/3	0/3
RMO 2000/3	0/1	0/1	0/1	0/1
RMO 2000/6	0/1	0/1	0/1	0/1
RMO 2001/2	0/1	0/1	0/1	0/1
Total	**9/16**	**6/16**	**17/28**	**16/28**

Table A1. *Complete supplied-set context. Each cell is successful runs / attempted runs. Denominators are shown because the two screens used different repetition counts.*

For the replication set, the paired table below makes policy overlap explicit: an A-only cell is one in which policy A passed and policy B failed at the same problem and repetition label.

A	B	A only	B only	Both	Neither	A-B
Q+Q silent	Q+G silent	9	3	20	0	+0.188
Q+Q silent	Q+G talking	10	3	19	0	+0.219
Q+G silent	Q+G talking	4	3	19	6	+0.031
Q+G silent	Qwen	9	4	14	5	+0.156

Table A2. *Paired outcomes on the 32 replication groups. A-B is the difference in final pass rate.*

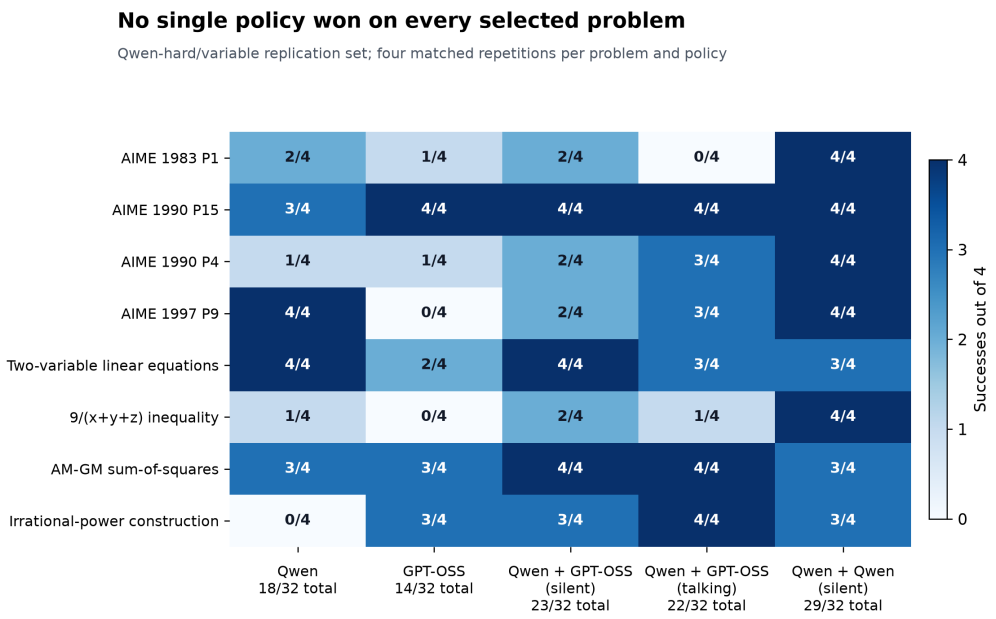

Figure A1. *Per-problem outcomes on the selected replication set. The selection was conditional on Qwen difficulty and is not representative of miniF2F as a whole.*

Appendix B. Measurement and reproducibility

The primary outcome is acceptance by a fresh run of the pinned grader-compatible Comparator. Warm Lean acceptance guides search but is not counted as final success. Problem-repetition labels pair conditions for analysis; provider sampling remained nondeterministic.

Evidence item	Committed source
Results data lock	f12c309fd91700c7ca2541d867dd4d1443b31171
Packet-variant data lock	c603da674f0c57323a38d5368ce49b02ddb7efb7
Supplied-set primary cells	96
Stage 6 result artifacts	160/160
Earlier replay	47 states; 752 responses
Replication replay	26 states; 416 responses

Table B1. *Data provenance for the regenerated tables and figures.*

The reproducible builder is `experiments/analysis/rebuild_paper_assets.py` in the submission bundle. It reads artifacts directly from Git refs, excludes the four Stage 5 pilots by requiring complete four-arm blocks, and emits the tables, figures, source-data CSVs, and a machine-readable manifest used here.

Appendix C. Development decisions

Observation	Design response	Role in final report
Single samples were brittle	Add compiler-guided repair and restart	Shared substrate for every arm
Whole failed proofs were noisy	Send only warm-compiling partial proofs	Motivated final message format
Messages often arrived after Qwen stopped	Reserve one Qwen call in both mixed arms	Keeps talking-vs-silent comparison matched
Marginal solo rates did not settle allocation	Add a real Qwen+Qwen policy	Separates mixed-model coverage from the complete allocation decision

Table C1. *Development observations that changed the final experimental design. Pilot outcomes are not pooled with the primary comparisons.*

Appendix D. Tools and assistance disclosure

OpenAI Codex and Claude assisted with implementation, orchestration, analysis, figures, and drafting using repository access. The work was conducted under continuous human supervision: the author repeatedly inspected intermediate outputs, requested explanations, redirected or interrupted agent work, overrode proposed decisions, and revised both the experimental and narrative direction. Agents had greater autonomy over individual coding operations and first-pass prose, but the author remained closely involved in drafting and retained responsibility for the research question, experimental conditions, promotion gates, interpretations, claims, and final text.