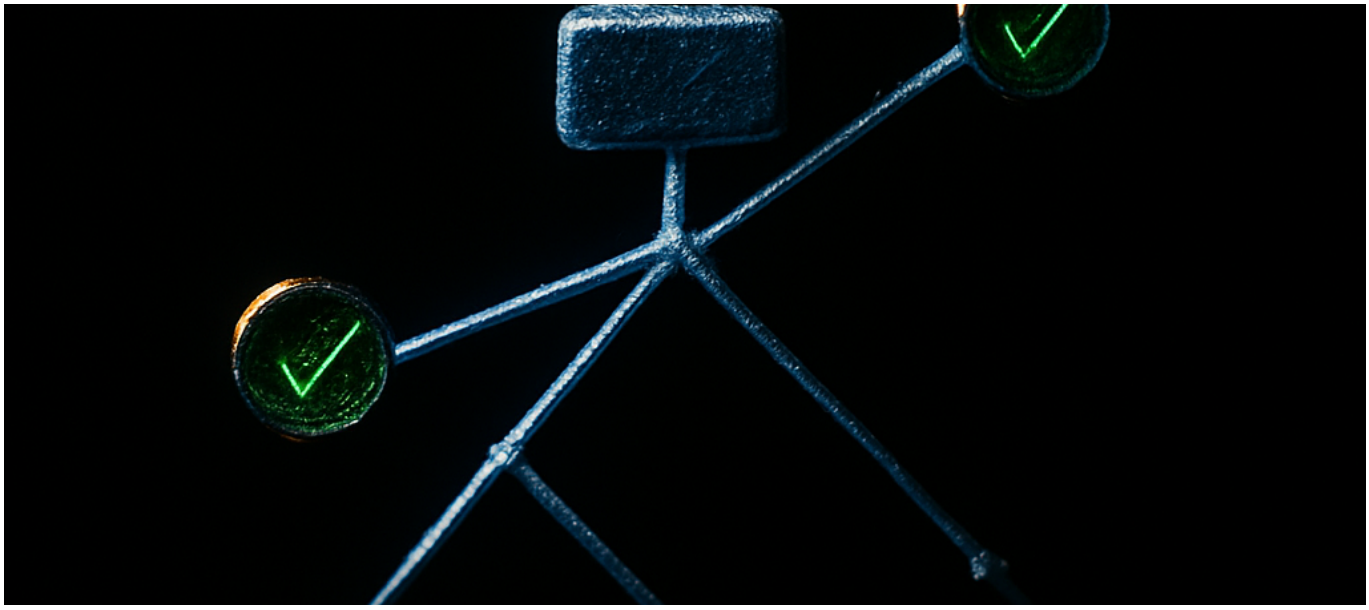

Mistral Releases Leanstral 1.5—119B Open-Source Model Hits 100% on miniF2F, Solves 587 of 672 Putnam Problems

Mistral’s new theorem prover found 5 unreported bugs across 57 repositories by proving code mathematically wrong—including an integer overflow hiding in production Rust. Verification just became cheaper than debugging.

The News: Mistral Saturates a Benchmark Most Thought Years Away

On July 2, 2026, Mistral AI released Leanstral 1.5, a 119-billion-parameter mixture-of-experts model that became the first open-source system to achieve 100% accuracy on the miniF2F formal mathematics benchmark. The model solved 587 of 672 PutnamBench problems—an 87.4% success rate on competition mathematics problems that historically stump most human undergraduates.

What makes this release different from typical benchmark-chasing announcements: Mistral open-sourced the entire model under Apache 2.0. It's available now on Hugging Face and through a free API. No enterprise contracts, no waitlists, no "contact sales for pricing."

The architecture uses 128 experts with 4 active per token, meaning only 6-6.5 billion parameters fire during inference despite the 119B total. This makes deployment practical—you get frontier theorem-proving capability at mid-tier model compute costs. The 256,000 token context window handles extended proof sessions where the model iterates within the Lean 4 compiler loop, proposing proofs and refining them until they verify.

For the benchmarks that matter to engineering teams: FLTEval pass@1 jumped from 21.9% to 28.9% compared to the previous Leanstral release. Pass@8 improved from 31.9% to 43.2%. On FATE-H (hard algebra), the model scores 87%. On FATE-X (extremely hard algebra), it manages 34%.

These aren't synthetic benchmarks. FATE-X problems represent the kind of mathematical reasoning required to prove properties about distributed systems, cryptographic primitives, and concurrent data structures.

Why This Matters: The Economics of Correctness Just Flipped

Formal verification has existed for decades. The reason your organization doesn't use it isn't technical—it's economic. Training engineers in Coq or Isabelle takes months. Writing proofs takes longer than writing code. The specialists who can do both command salaries that make verification prohibitively expensive for anything but safety-critical aerospace and medical systems.

Leanstral 1.5 changes the cost function. When a model can iterate through proof attempts automatically—scaling from 44 solved problems at 50k tokens to 587 at 4M tokens on PutnamBench—the bottleneck shifts from human expertise to compute. Compute gets cheaper every year. Human formal methods experts remain scarce.

The 5 bugs discovered across 57 repositories in Mistral's testing illustrate concrete value. One was an integer overflow in a Rust library triggered when a value

incremented past `Std.U64.MAX`. This is exactly the class of bug that unit tests miss, fuzzers miss, and code review misses—but mathematical proof catches deterministically.

The winners: Small teams building high-reliability systems. Fintech companies that previously couldn't afford formal verification for smart contracts. Any organization with compliance requirements that mandate provable correctness.

The losers: Proprietary formal verification tool vendors charging six figures annually. Consultancies whose value proposition rested on scarce expertise. Organizations that bet heavily on less rigorous testing approaches now facing competitive pressure from teams shipping proven code.

The aerospace and defense sectors—historically the only industries that could justify formal verification costs—now face a different problem: their competitors outside regulated industries can match their correctness guarantees at a fraction of the cost.

Technical Depth: How Leanstral 1.5 Actually Works

Understanding why this model succeeds where previous attempts failed requires examining three architectural decisions: the mixture-of-experts structure, the compiler-in-the-loop training, and the test-time scaling behavior.

Mixture-of-Experts for Proof Search

The 128-expert architecture with 4 active per token isn't just parameter efficiency—it's functional specialization. Different proof strategies require different reasoning patterns. Induction proofs differ fundamentally from case analysis. Algebraic manipulation differs from topological arguments.

By routing different proof steps to different expert combinations, the model can maintain diverse proof strategies without interference. A standard dense transformer would blend these approaches into a single compromised representation. The MoE structure keeps them distinct.

The 6-6.5B active parameters per token put inference costs in range of GPT-4 class

models while total capacity exceeds most competitors. This matters for deployment: you can run meaningful proof sessions on hardware that doesn't require a datacenter allocation.

Compiler-in-the-Loop Operation

Leanstral 1.5 operates within the Lean 4 compiler loop, not as an external oracle that generates proofs blindly. The model proposes a proof step, the Lean compiler verifies it or returns an error, and the model incorporates that feedback into its next attempt.

This tight integration means the model never hallucinates proofs. Every step either verifies or fails explicitly. The 256k token context window accommodates extended sessions where initial strategies fail and the model must backtrack significantly—Mistral recommends up to 200k tokens for long agentic sessions where complex proofs require multiple strategy pivots.

The Lean 4 choice matters strategically. Lean 4 has the fastest-growing formal verification community, the most active library development (Mathlib), and the most momentum among new practitioners. Building for Lean 4 rather than Coq or Isabelle bets on where the ecosystem is heading, not where it was.

Test-Time Scaling Reveals True Capability

The PutnamBench pass@8 results at different token budgets tell the real story of this model's capability:

- 50k tokens: 44 problems solved
- 200k tokens: 244 problems solved
- 1M tokens: 493 problems solved
- 4M tokens: 587 problems solved

This scaling curve means compute substitutes directly for capability. If you need higher success rates, you spend more tokens. If you're proving simpler properties, you can run cheaper sessions. The relationship is predictable enough to inform engineering decisions about where formal verification provides positive ROI.

The 100% miniF2F score saturates that benchmark entirely. MiniF2F was designed as a challenge dataset—problems drawn from mathematical olympiads that require

multi-step reasoning and creative proof strategies. Saturating it doesn’t mean the model has solved mathematics; it means the benchmark no longer discriminates between models at the frontier. The community will need harder benchmarks.

The Contrarian Take: What the Coverage Gets Wrong

Most coverage of Leanstral 1.5 focuses on the benchmark numbers and the open-source angle. Both are real achievements, but they obscure the more interesting story.

The Overhyped Part: Benchmark Saturation

Hitting 100% on miniF2F sounds like the model has mastered formal mathematics. It hasn’t. MiniF2F contains 488 problems—significant, but representing a narrow slice of mathematical reasoning. The benchmark was created in 2021 and has been a target for improvement ever since. Saturating it means the benchmark exhausted its discriminative power, not that the underlying problem is solved.

The 87.4% on PutnamBench is more informative precisely because it’s not 100%. The remaining 12.6% of problems—85 competition math challenges the model cannot solve—tell us where the capability ceiling actually sits. These unsolved problems likely share characteristics that reveal fundamental limitations in current approaches.

Similarly, the 34% on FATE-X shows that extremely hard algebraic reasoning remains difficult. For practical engineering applications, this matters less—few verification tasks require olympiad-level algebraic insight. But it prevents the narrative that AI has “solved” mathematical reasoning.

The Underhyped Part: The Bug Discovery Methodology

Five bugs across 57 repositories barely registered in the coverage. This should have been the headline.

Consider what this means operationally: Mistral ran Leanstral 1.5 against existing codebases—production Rust libraries with active maintainers—and found integer overflow bugs that escaped all existing quality assurance. These weren’t toy

examples or adversarial inputs. They were real bugs in real code that could have caused real failures.

The methodology for systematic bug hunting through formal verification at scale is more valuable than any benchmark score. When you can point an AI at a codebase and have it mathematically prove the presence of bugs, you've changed the software development feedback loop fundamentally.

This capability compounds. Each verified property becomes a regression test that can never pass falsely. Each proven invariant constrains future modifications to remain correct by construction. The economic value of one prevented production incident typically exceeds the cost of extensive verification by orders of magnitude.

What Everyone Ignores: The Training Data Question

The announcement doesn't detail what proofs Leanstral 1.5 trained on. This matters for organizations considering adoption.

Lean 4's Mathlib contains extensive mathematical proofs, but engineering-relevant verification—memory safety properties, concurrent program correctness, cryptographic protocol analysis—requires different training distributions. A model optimized for pure mathematics may underperform on software verification tasks despite strong benchmark scores.

Organizations should benchmark Leanstral 1.5 against their specific verification needs before committing. The model that proves olympiad problems isn't necessarily the model that proves your authentication logic correct.

Practical Implications: What Engineering Leaders Should Do Now

Leanstral 1.5 is available today under Apache 2.0. The question isn't whether to pay attention—it's what concrete actions make sense for different organizational contexts.

If You Have Zero Formal Verification Experience

Start with the free API, not local deployment. Mistral provides API access that lets

you experiment without infrastructure investment.

Pick one critical function in your codebase—ideally something with clear correctness requirements like a parser, a state machine, or an authentication check. Write a Lean 4 specification for what that function should do. Point Leanstral 1.5 at proving the implementation matches the specification.

You'll likely fail the first several attempts. The value is in discovering where your mental model of the code differs from its actual behavior. Even failed proof attempts surface assumptions you didn't know you were making.

Allocate one engineer for one sprint to this experiment. The goal isn't production deployment—it's building organizational intuition for where formal verification adds value in your specific context.

If You Already Use Formal Methods

Your competitive advantage just eroded significantly. The expertise you've built remains valuable—knowing what to prove matters as much as proving it—but the execution barrier that protected your investment has lowered for everyone.

Integrate Leanstral 1.5 into your existing workflows as an acceleration tool. Let the model handle routine proof obligations while your experts focus on specification design and strategy. Measure the productivity gain; Mistral's numbers suggest pass rates roughly double compared to the previous version on equivalent tasks.

Evaluate whether your proprietary formal verification investments still provide positive ROI. If you're paying significant licensing fees for commercial proof assistants, the Apache 2.0 model may eliminate that cost entirely.

If You're Building Developer Tools

The IDE integration opportunity is massive. Formal verification can now reach developers who've never heard of Lean.

Imagine: a developer writes a function, an AI generates a specification based on the function's apparent intent, another AI (or the same one) proves whether the implementation satisfies that specification, and the results surface as inline warnings in the editor. No formal methods training required.

The workflow looks like enhanced type checking—something developers already understand. The implementation requires stitching together specification inference, proof generation, and developer experience. Each of these is now tractable.

The teams that build this developer experience layer will capture significant value. The underlying model is free; the integration and UX are not.

Code to Try Today

If you want to experiment immediately, here's the minimal path:

1. Install Lean 4 and its toolchain (elan)
2. Access Leanstral 1.5 via Hugging Face or Mistral's API
3. Start with Mathlib's existing proven theorems as templates
4. Write specifications for your own code following those patterns
5. Run proof attempts with increasing token budgets until success or clear failure

The recommended workflow uses up to 200k tokens for complex proofs. Start with 50k token sessions to understand the model's behavior before scaling compute.

Forward Look: What Happens in the Next 12 Months

Leanstral 1.5's release accelerates several trends that will reshape software development practices.

Formal Verification Becomes a CI/CD Standard

Within six months, expect CI/CD integrations that run formal verification alongside traditional tests. The workflow: on every pull request, prove that new code doesn't violate existing verified properties. If the proof fails, block the merge.

This creates an asymmetric quality guarantee. Once a property is proven, it cannot regress—any change that would violate it fails the proof check before reaching production. Traditional test suites can only check behaviors they anticipate; formal verification checks all behaviors within its specification scope.

GitHub, GitLab, and the major CI platforms will race to offer native support. The

platforms that integrate fastest will attract teams building high-reliability systems.

Specification Languages Become a Hiring Skill

The bottleneck in formal verification is no longer proof construction—it's specification authoring. Models can prove properties, but they cannot decide which properties matter.

Writing good specifications requires understanding both the business domain (what should this code accomplish?) and the formal domain (how do we express that requirement precisely?). This skill combines software architecture, logic, and domain expertise in ways that resist automation.

Job postings in 12 months will list Lean 4 alongside TypeScript. The engineers who can write specifications that capture meaningful properties will command premium compensation.

The Insurance and Compliance Markets Notice

Cyber insurance underwriters currently price policies based on observable security practices—penetration testing, SOC 2 compliance, incident response plans. Formal verification provides stronger guarantees than any of these approaches, but hasn't been practical to require.

Leanstral 1.5 changes the cost calculus. If you can prove your authentication logic correct rather than merely testing it, you've eliminated a class of vulnerabilities entirely. Insurers will notice. Compliance frameworks will update.

Expect early movers in fintech and healthcare to seek competitive advantage by demonstrating formally verified properties in their security posture. "Our payment processing code is mathematically proven correct" becomes a sales differentiator.

Other Labs Will Follow Fast

OpenAI, Anthropic, and Google have all demonstrated formal reasoning capability. Mistral's open-source release forces their hand. The competitive dynamic that made GPT-4 proprietary doesn't apply to theorem provers—formal verification benefits from transparency, and restricting access limits the user base that drives improvement.

Expect competing open-source formal verification models within the year. This is good for practitioners: competition drives capability improvement, and Apache 2.0 licensing means the ecosystem builds rather than fragments.

The risk is benchmark gaming. As miniF2F saturates and new benchmarks emerge, labs may optimize for leaderboard position rather than practical verification utility. Engineers adopting these tools should benchmark against their own use cases, not published numbers.

Hardware Vendors Adjust

The test-time scaling characteristics of Leanstral 1.5—where token budget directly converts to success rate—create demand for inference infrastructure optimized for long-context, high-throughput sessions.

Current GPU architectures optimize for training throughput or low-latency inference. Formal verification demands neither; it demands extended reasoning sessions where the model iterates for minutes or hours toward a proof. Memory bandwidth matters more than raw FLOPS.

Expect inference hardware announcements tailored to “reasoning workloads” by year end. The economics of proving code correct at scale depend on hardware that matches the workload.

What This Means for Your Strategy

Leanstral 1.5 represents an inflection point in the practical accessibility of formal verification. Not because the underlying mathematics changed, but because the human expertise barrier dropped dramatically.

For most engineering organizations, the immediate action is experimentation: try the model on a contained problem, learn the workflow, assess fit. The cost is near zero; the potential value is significant.

For organizations already invested in software quality, the question is integration: how does AI-assisted formal verification fit into existing practices? Where does it augment testing? Where does it replace it? What new guarantees become achievable?

For tool builders, the opportunity is interface design: the model exists, but the developer experience doesn’t. Whoever builds the IDE integration that makes verification feel as natural as syntax highlighting will capture substantial market share.

The underlying shift is economic. Proving code correct was always possible in principle. Now it’s affordable in practice. That changes which bugs are acceptable, which systems can claim reliability, and which teams can compete on correctness.

The organizations that integrate AI-assisted formal verification into their development workflows over the next 12 months will establish a quality advantage that compounds with every proven property, every prevented bug, and every system that works correctly by construction rather than by chance.