Multi-Agent Strategy for Solving Research-Level Mathematics

Documentation of the Workflow Used in the “First Proof” Benchmark

Dietmar Wolz

February 9, 2026

Abstract

This document details a heterogeneous multi-agent workflow designed to solve research-level mathematical questions, specifically those presented in the “First Proof” benchmark (arXiv:2602.05192v1). The strategy leverages the distinct strengths of three frontier models: **Claude Opus 4.6** for computational verification and code optimization, **Gemini 3.0 Pro** for architectural planning and proof synthesis, and **ChatGPT 5.2 (Extended Thinking)** for adversarial review. The methodology employs a conditional branching initialization followed by an iterative agentic refinement loop. A fuller design rationale and implementation notes are provided in the companion design document [2].

1 Introduction

The “First Proof” benchmark presents ten research-level questions arising from fields such as algebraic combinatorics, spectral graph theory, and stochastic analysis [1]. Solving these requires more than simple retrieval; it requires hypothesis testing, rigorous formalization, and peer-review-level scrutiny.

This document outlines a standardized protocol for solving these problems by assigning specific roles to AI agents based on their respective strengths and typical failure modes:

- **Computational Agent:** Opus 4.6 (Code generation, optimization, numerical heuristics).
- **Synthesis Agent:** Gemini 3.0 Pro (Thinking/Reasoning, LaTeX drafting, workflow orchestration).
- **Review Agent:** ChatGPT 5.2 (Extended Thinking, logical auditing).
- **Formal Proof Checker (selective):** Aristotle (Harmonic) for Lean-level verification and counterexample feedback [3].

A more detailed design specification of this workflow (including prompt templates and failure-mode mitigations) is maintained separately in [2].

2 Workflow Architecture

The solution pipeline is determined by the nature of the problem. If the problem implies a need for numerical verification or counterexample search, the *Computational Branch* is triggered. Otherwise, the process begins directly with the *Theoretical Branch*.

2.1 Phase 1: Initialization and Hypothesis Generation

Role: Claude Opus 4.6 – The Experimentalist

Trigger Condition: Problem requires numerical search, pattern recognition, or stochastic simulation.

Task 1: Experimental Coding

- **Input:** The raw problem statement from the PDF.
- **Action:** Create initial Python scripts to simulate the mathematical objects (e.g., tensor decompositions, Markov chains, or polynomial roots).

Task 2: Code Optimization

- **Input:** Initial Python script.
- **Action:** Refine the code for high-performance execution.
- **Specific Techniques:**
  - Implementation of `multiprocessing` (specifically `imap`) for parallel execution.
  - Implementation of `numba` (JIT compilation) to speed up inner mathematical loops.

Task 3: Proof Sketch

- **Action:** Analyze the numerical output (convergence rates, counterexamples, or invariant patterns) and generate a natural language proof sketch or conjecture based on the empirical data.

2.2 Phase 2: Architectural Planning and Synthesis

Role: Gemini 3.0 Pro – The Architect & Author

Trigger Condition: Always active. Receives input from Phase 1 (if applicable) or starts here.

Task 1: Context Ingestion

- **Input:**
  - Full PDF context of the question.
  - (Optional) Proof sketch, code, and code output from Opus 4.6.

Task 2: Meta-Prompting (the "Plan")

- **Action:** Before solving, create a comprehensive prompt plan.
- **Goal:** Identify necessary lemmas, definitions, and the logical order of operations required for a formal proof.
- **Output:** A structured "Table of Contents" for the proof.

Task 3: Execution (First Draft)

- **Action:** Execute the plan generated in Task 2.
- **Output:** A comprehensive, formal proof written in LaTeX.

2.3 Phase 3: The Agentic Review Loop

This phase constitutes the core rigorous element of the strategy. It is an iterative loop between the *Review Agent* and the *Synthesis Agent*.

Algorithm 1 Agentic Review Loop

```
1: Input: $P_0$ (Initial Proof from Gemini)
2: $i \leftarrow 0$
3: $Satisfied \leftarrow$ False
4: while $Satisfied$ is False do
5:     ChatGPT 5.2: $R_i \leftarrow \text{Review}(P_i)$
6:     if $R_i$ contains no critical errors then
7:         $Satisfied \leftarrow$ True
8:     else
9:         Gemini 3.0 Pro: $P_{i+1} \leftarrow \text{UpdateProof}(P_i, R_i)$
10:        $i \leftarrow i + 1$
11:    end if
12: end while
13: Output: Final Proof $P_{final}$
```

Agent Roles in the Loop

Role: ChatGPT 5.2 (Extended Thinking) – The Reviewer

Task: Adversarial Review

- **Prompt:** "Review the attached revised proof of question X. Scrutinize logical leaps, citation accuracy, and algebraic correctness. Create a downloadable LaTeX file of the review."
- **Mechanism:** Uses "Extended Thinking" to simulate the time-intensive peer-review process, checking step-by-step derivations.
- **Output:** A `.tex` file containing specific critiques, counter-claims, or requests for clarification.

Role: Aristotle (Harmonic) – The Formal Proof Checker

Trigger Condition: Used selectively when a key lemma or technical step could be formalized in Lean 4, or when a natural-language argument passed the LLM review but still benefited from machine-checking.

Task: Formal Verification / Hole Filling

- **Input:** A Lean proof skeleton (often created by translating a local LaTeX fragment) with explicit goals and required imports.
- **Action:** Use Aristotle to (i) fill in missing proof steps, (ii) surface typeclass/import issues, or (iii) produce counterexamples when the statement is false.
- **Output:** A Lean-verified proof (or a concrete failure mode) that is fed back into the review loop [3].

Role: Gemini 3.0 Pro – The Editor

Task: Refinement and Correction

- **Input:** The LaTeX review generated by ChatGPT 5.2.
- **Action:**
  - Check the validity of the claims made in the review.
  - Re-derive sections flagged as erroneous.
  - Improve the clarity of the arguments.
- **Output:** A revised version of the full proof in LaTeX.

3 Summary of Prompts Used

The following specific prompts dictate the interaction flow:

Opus 4.6 (Computational)

1. "Create experimental Python code to [simulate/solve] the following problem..."
2. "Improve this code using multiprocessing (imap) and numba to maximize execution speed."

3. “Based on the output, create a proof sketch or conjecture.”

Gemini 3.0 Pro (Synthesis)

1. **Planning:** “Create a prompt for the creation of a comprehensive proof in LaTeX. Break down what needs to be done and in which order.”
2. **Drafting:** “Follow the prompt plan created above and generate the comprehensive proof in LaTeX.”
3. **Updating:** “Check the claims in the attached review. Improve the proof if necessary and output the updated LaTeX.”

ChatGPT 5.2 (Review)

1. “Review the attached revised proof of question X. Identify any logical fallacies or gaps. Create a downloadable LaTeX file of the review.”

4 Improvements

1. Add an additional create-code / review-code loop for experimental scripts (separate from the proof loop).
2. Constrain each review iteration to a fixed “change budget” (e.g., three high-impact improvements) to keep revisions incremental and auditable.
3. Require the reviewer to propose concrete replacement text (ready to paste into LaTeX), and require the editor to explicitly accept or reject each proposal.

Prompt Templates for the Review Loop

Template A: Targeted improvement review

```
Check the revised proof.  Propose the three (more) most important improvements /
refinements of the proof.  Make proposals what to write exactly.  Create LaTeX for
this review.
```

Template B: Review-to-revision loop (accept/reject each point)

```
Check the review and improve the proof for each point step for step, but only
when you agree.
The review contains the three most important things to improve and proposes
changes you can review/use.
If something needs to be derived instead of being just asserted, do so.  Create
downloadable LaTeX.
```

5 Conclusion

This methodology cleanly separates concerns: Opus handles computational experiments and performance engineering, Gemini handles proof architecture and LaTeX authoring, and ChatGPT provides adversarial, peer-review–style scrutiny as a “second pair of eyes.” For selected proofs, we additionally used the Aristotle system as a formal Lean 4 proof checker, providing a machine-verifiable backstop on key lemmas and technical steps [3]. This reduces ungrounded assertions and improves both correctness and readability of the final mathematical output.

References

[1] Abouzaid, M., et al. (2026). *First Proof*. arXiv:2602.05192v1 [cs.AI].

[2] Wolz, D. (2026). *Agentic Strategy Design for Math Proofs*. Available at https://althofer.de/agentic_strategy_design_for_math_proofs.pdf. Accessed February 12, 2026.

[3] Achim, T., et al. (The Harmonic Team) (2025). *Aristotle: IMO-level Automated Theorem Proving*. arXiv:2510.01346. Project page: https://harmonic.fun. Accessed February 12, 2026.