The Agentic Researcher: A Practical Guide to AI-Assisted Research in Mathematics and Machine Learning

Max Zimmer [1 2] **Nico Pelleriti** [1 2] **Christophe Roux** [1 2] **Sebastian Pokutta** [1 2]

Abstract

AI tools and agents are reshaping how researchers work, from proving theorems to training neural networks. Yet for many, it remains unclear how these tools fit into everyday research practice. This paper is a *practical guide to AI-assisted research in mathematics and machine learning*: We discuss how researchers can use modern AI systems productively, where these systems help most, and what kinds of guardrails are needed to use them responsibly. It is organized into three parts: (I) a five-level taxonomy of AI integration, (II) an open-source framework that, through a set of methodological rules formulated as agent prompts, turns CLI coding agents (e.g., Claude Code, Codex CLI, OpenCode) into autonomous research assistants, and (III) case studies from deep learning and mathematics. The framework runs inside a sandboxed container, works with any frontier LLM through existing CLI agents, is simple enough to install and use within minutes, and scales from personal-laptop prototyping to multi-node, multi-GPU experimentation across compute clusters. In practice, our longest autonomous session ran for over 20 hours, dispatching independent experiments across multiple nodes without human intervention. We stress that our framework is not intended to replace the researcher in the loop, but to augment them.

1. Introduction

In 2024, DeepMind's AlphaProof (Hubert et al., 2025) combined with AlphaGeometry (Trinh et al., 2024) became the first AI system to achieve medal-level performance at the International Mathematical Olympiad (IMO), reaching silver-medal standard by solving four of the six competition problems through reinforcement learning and formal verification. AlphaEvolve (Novikov et al., 2025) demonstrated that LLM-guided evolutionary search can discover new mathematical constructions, rediscovering best-known solutions across a broad collection of problems and improving on them in several cases (Georgiev et al., 2025). Most recently, Aletheia (Feng et al., 2026b), an autonomous mathematical research agent, resolved several open problems originally posed by Erdős while operating with minimal human intervention. Aletheia also solved several open problems from *First Proof* (Abouzaid et al., 2026), a benchmark of previously unpublished research-level mathematics questions drawn from the authors' own research process, within *weeks* of its release. These results are remarkable, and recent systems now address not only well-defined benchmarks but also genuine open mathematical problems. In parallel, the Machine Learning (ML) community has seen a surge in agentic experimentation: for instance, Karpathy's *autoresearch* (Karpathy, 2026) demonstrated how agents can run automated ML experiment pipelines through iterative code modification, and such pipelines are becoming increasingly common.

Most of this literature focuses on *what AI systems can achieve*; far less attention has been paid to *how individual researchers should integrate such systems* into their daily work. Research rarely proceeds along a fixed objective: researchers decide which questions to ask, when to reformulate a conjecture, and how to respond to unexpected results. Supporting this requires workflows that accommodate shifting objectives, iterative experimentation, and sustained human guidance. Conceptual proposals abound (Haase & Pokutta, 2026; Henkel, 2025; Avigad, 2026), but none gives end-to-end guidance a researcher can follow today.

We hope to make *some* progress on these questions and aim to fill *parts of* that gap. We present (i) a practical taxonomy of AI integration into mathematical and ML research; (ii) an open-source, sandboxed framework that wraps general-purpose CLI coding agents with a small set of methodological rules—the *Ten Commandments*—and reporting conventions; and (iii) case studies from deep learning and math-

[1]Department for AI in Society, Science, and Technology, Zuse Institute Berlin, Germany [2]Institute of Mathematics, Technische Universität Berlin, Germany. Correspondence to: Max Zimmer <zimmer@zib.de>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

Table 1. Five levels of AI integration in mathematical research. Each level represents a different trade-off between agent autonomy and human involvement.

Level	Name	Typical use
0	Classical	No AI integration; traditional LaTeX, mathematical software, and custom code.
1	Consultant	Targeted chatbot queries for explanation, literature search, brainstorming, or proof debugging.
2	Typist	Editor plugins generate code or text, but do not execute, test, or iterate.
3	Collaborator	CLI coding agents read files, edit code, run commands, and iterate on a task assigned by the human.
4	Research Assoc.	An autonomous experiment loop follows a research plan, records results, and requests input only when needed.

ematics. The main text focuses on four mathematical and scientific-computing case studies: a lower-bound proof in convex optimization, a proof-and-implementation workflow for mixed-integer optimization, weight reconstruction for LLM pruning, and column ordering in LLM quantization. Throughout, the agent operates inside a container and the researcher remains the principal investigator, steering the process and verifying results.

2. Levels of AI Integration

Inspired by Haase & Pokutta (2026), we use a five-level taxonomy to distinguish qualitatively different ways AI enters the research process (Table 1). These levels are not mutually exclusive: a single project may combine a chatbot for literature triage, a code-completion system for boilerplate, a CLI agent for implementation, and a higher-level autonomous loop for sustained exploration. The key practical question is not whether a researcher "uses AI," but which level is appropriate for which part of the workflow and how much verification the delegated work requires.

Levels 0–2: local assistance. At the lowest levels, the researcher remains responsible for all execution and design decisions. Chatbots can explain a definition, suggest related work, or critique a proof attempt, while editor plugins can accelerate routine coding and writing. These modes are useful, but their limitations are structural: the AI does not maintain durable project state, does not run experiments, and does not close the loop between a mathematical idea, an implementation, and an observed failure.

Level 3: interactive delegation. CLI coding agents change the interaction pattern. A researcher can ask for a Frank–Wolfe implementation, a Lean or Julia verification script, or a reproduction of a numerical experiment; the agent reads the repository, edits files, runs commands, and debugs errors. This is already a major shift for mathematical research because many proof attempts now have executable companions: symbolic checks, exhaustive enumeration on small instances, high-precision numerical experiments, and regression tests can be produced in the same session as the derivation.

Level 4: autonomous research loops. The highest level is the target of our framework. The researcher provides a research idea, constraints, evaluation criteria, prior work, and relevant code or notes. The agent then executes a loop: formalize the current question, plan the next step, implement or derive, evaluate, analyze, record, and iterate. Human review is still central. The researcher steers the project, audits claims, and decides what results matter. The agent's role is to externalize sustained exploratory labor while leaving a written trail that can be inspected and corrected.

3. The Agentic Research Framework

Inputs. To start a project, the researcher provides three things (Figure 1): a *research question* (problem formulation, hypotheses, evaluation criteria), the *tools, methods, and data* needed (software stack, datasets, compute), and any *prior work or domain knowledge* that should inform the investigation (codebase, LaTeX derivations, references, partial results). The framework[1] is built around CLI coding agents (e.g., Claude Code (Anthropic), Codex CLI (OpenAI), OpenCode (Anomaly), Gemini CLI (Google)) operating inside a sandboxed container that confines all actions to an isolated workspace.

Workflow. A session proceeds as follows: (1) the researcher launches the sandbox in a project directory containing the materials above and provides natural-language research instructions; (2) the agent asks clarifying questions about scope, constraints, and metrics; (3) the agent writes the final project-specific instructions into a persistent `INSTRUCTIONS.md` alongside the universal commandments (Section 3); (4) the agent initializes `report.tex` and `TODO.md`, drafts a plan, and—after approval—executes autonomously, requesting human input only on unexpected behavior or when the plan needs revision.

Sandbox and tools. The agent operates inside an Apptainer container that confines all actions to three known locations: a read-write `/workspace` containing the project, an isolated home directory holding agent state and credentials, and a `/scratch` space for large artifacts such as checkpoints, datasets, and logs (the project directory has limited disk and is kept small). The host filesystem and other projects are inaccessible. Inside the container, the agent has the devel-

[1]Code: github.com/ZIB-IOL/The-Agentic-Researcher

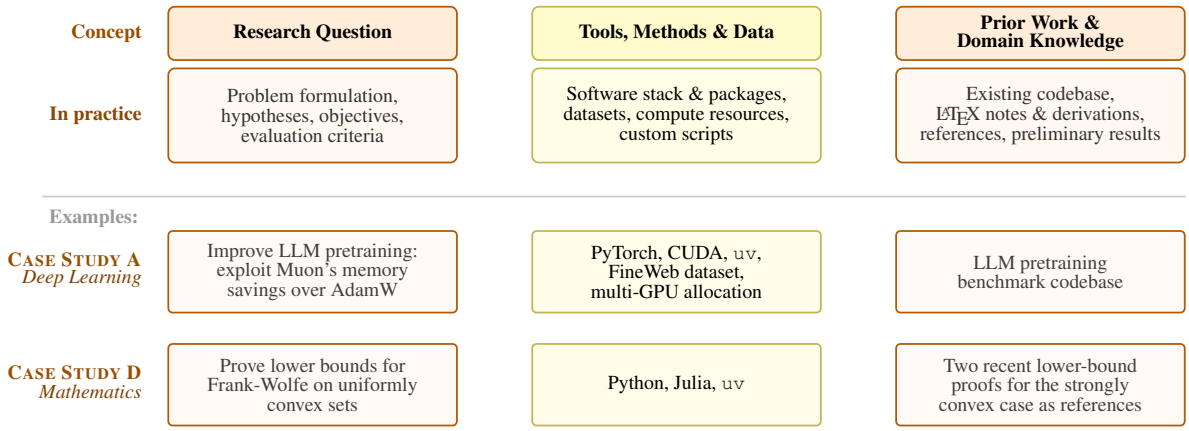

Figure 1. Setting up a research project. **Top:** the three categories of input the researcher provides, with their conceptual role (dark) and concrete realization (light). **Bottom:** two examples drawn from our case studies: a deep-learning project and a mathematics project.

oper tools needed to run a research project end-to-end: `git` and `gh` for version control and pull requests, `ripgrep`, `jq`, and `yq` for searching code and structured data, `curl` and `wget` (proxy-routed) for fetching papers from arXiv and external resources, Python via `uv` for fully reproducible environments, Julia, and a TeX Live distribution used in read/edit mode only. On GPU machines, the agent discovers devices with `nvidia-smi` and pins independent experiments to specific GPUs via `CUDA_VISIBLE_DEVICES` (Commandment C1); under a multi-node Slurm allocation, a small dispatcher (`remote-run`) writes job files that a host-side daemon executes in the same container image on remote nodes, so the agent on the head node can asynchronously fire fully independent experiments to other nodes while continuing local work.

Structured reporting. All experimental progress is recorded in a single LaTeX file (`report.tex`), which accumulates experiments, derivations, and analysis. A `TODO.md` checklist captures open questions, unverified claims, and deferred work. Each experiment subsection must contain the following fields, enforced by the commandments:

```
\paragraph{Goal} What problem are we
    solving?
\paragraph{Hypothesis} Why should this
    approach work?
\paragraph{Method} Mathematical
    formulation
   with proper notation.
\paragraph{Implementation} Files and lines
   changed.
\paragraph{Results} Table with method,
    model
   /instance, metric, delta.
\paragraph{Analysis} Why it worked or didn
    't.
\paragraph{Next Steps} What to try next.
```

Rather than introducing a separate experiment-tracking system, we use Git directly: each experiment is recorded as a commit with a structured message of the form `exp(EXXX): <description> -- <metric>=<value>`, and Git's worktree feature lets multiple agent sessions run concurrently on separate copies of the codebase. Once running, each experiment follows the eight-step loop shown in Figure 2: *Explore* → *Plan* → *Implement* → *Evaluate* → *Analyze* → *Record* → *Commit* → *Iterate*. At session start (and after any context-window reset), the agent re-reads `report.tex`, `TODO.md`, and the `git log` to restore continuity.

The Ten Commandments. The core of the framework is a small set of rules distilled from observed failure modes, each formulated as a falsifiable directive in the agent's instruction file: (I) Never break a promise; (II) Never manipulate evaluation; (III) Never fabricate citations; (IV) Complete all autonomous work before reporting; (V) Make it work before moving on; (VI) One variable per experiment; (VII) Evaluate in tiers (sanity-check → subset → full); (VIII) Bound your expectations (compute an oracle / theoretical limit before chasing heuristics); (IX) Record everything (in `report.tex` and `TODO.md`; if it is not in the report, it did not happen); (X) Verify before claiming (write verification scripts; actively try to falsify your own claims). Domain-specific addenda govern compute-intensive work (multi-GPU dispatch, memory, context hygiene) and mathematical research (precise notation, derivations-before-code, counterexample-first reasoning). Full text, failure modes that motivated each rule, and domain modules appear in Appendix C. The full five-level taxonomy of AI integration that situates this workflow (Levels 0–4, from classical to research-associate) is given in Appendix A.

Why these rules matter for mathematical research. For mathematical workflows, the most important distinction is between a plausible narrative and a checked artifact. The

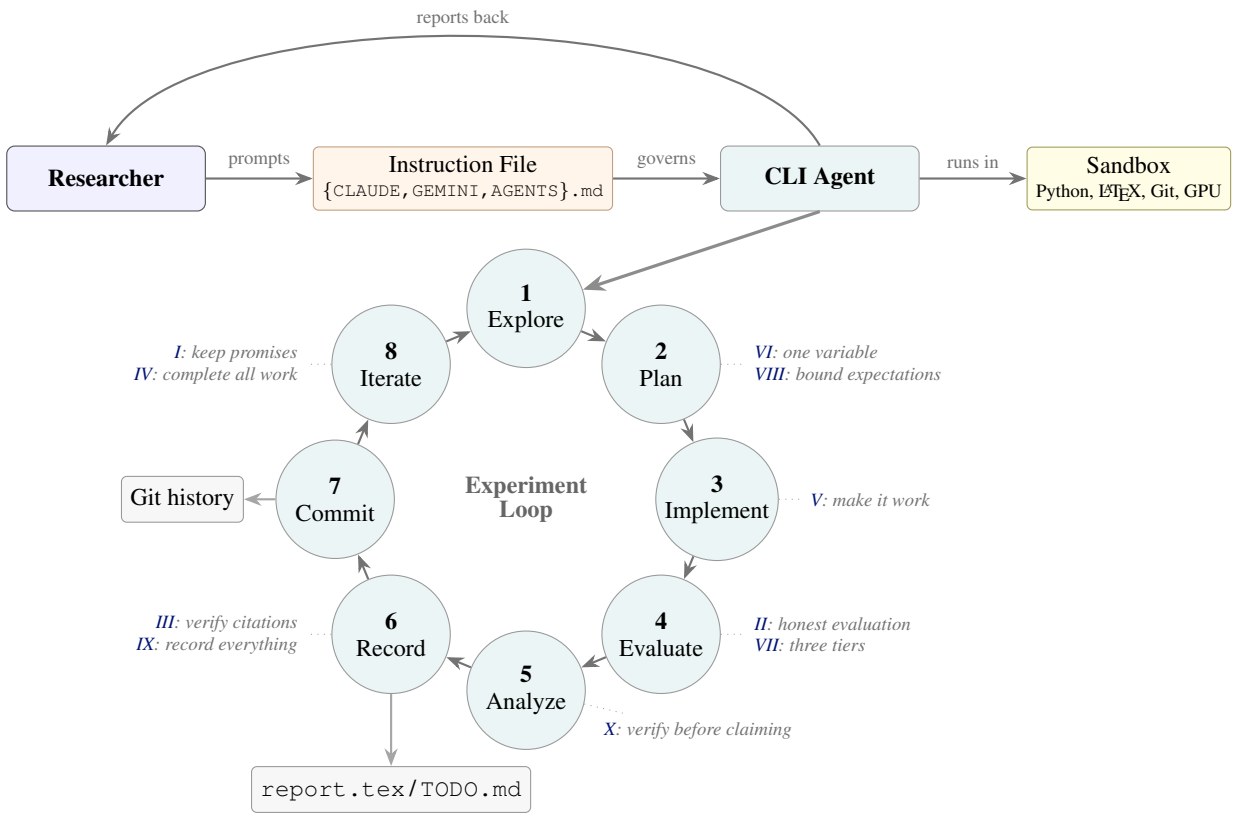

Figure 2. Overview of the agentic research framework. **Top:** the researcher writes a persistent instruction file that governs the CLI agent operating within a sandboxed environment. **Bottom:** each experiment follows an eight-step loop, with every step explicitly governed by one or more of the Ten Commandments.

commandments force the agent to produce the latter whenever possible. "Verify before claiming" means that a derived recurrence should be checked against brute force on small instances, a proposed solver modification should preserve optimal objectives on controlled tests, and a computational discovery should be stress-tested under perturbations. "Record everything" makes negative results useful: failed generalizations, counterexamples, implementation bugs, and unstable numerical patterns become part of the research state rather than disappearing when the model context is compacted. "One variable per experiment" prevents a common failure in both empirical and mathematical exploration: when a method improves after several simultaneous changes, the agent cannot know which change mattered.

Mathematical modules. The general commandments are supplemented by domain-specific rules for proof-heavy work. First, *derivations before code*: before implementing a method, the agent must write the mathematical formulation, define notation, and cross-reference the source equations it is using. Second, *precise notation*: indices, dimensions, scalar/vector/matrix distinctions, and quantifier ranges must be explicit. Third, *counterexample-first reasoning*: before investing in a proof, the agent searches for counterexamples through random tests, boundary cases, and exhaustive enumeration on small instances. In our case studies, these rules did not merely catch mistakes; they often exposed the structure that made the successful proof or computation possible.

4. Case Studies

We illustrate the framework on four projects: two empirical ML studies and two mathematical studies. The remaining case studies from the long version, including the optimizer exploration and power-network search, are in Appendix E. In every figure shown here, the agent's report is reproduced verbatim with a thin border around the embedded artifact and [*sic*] markers on any minor rendering artifacts.

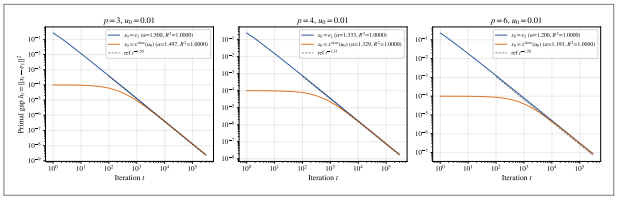

Figure 3. A plot [*sic*] from the agent's report: log–log convergence of $\|x_t - e_1\|^2$ for $p \in \{3, 4, 6\}$ starting from $x_0 = e_2$ (blue) and from the worst-case initialization $x_0^{\text{slow}}(10^{-2})$ from the proof (orange); α is the fitted coefficient of $t^{-\alpha}$.

4.1. Tight Lower Bounds for Frank–Wolfe on Uniformly Convex Sets

Setting. The Frank-Wolfe (FW) algorithm minimizes a smooth convex function over a convex constraint set using only a linear minimization oracle (LMO). On strongly convex sets, the known $\mathcal{O}(1/T^2)$ upper bound was recently shown to be tight: Halbey et al. (2026) gave a lower bound for vanilla FW in dimension 2 by analyzing the dynamics of the iterates on a worst-case instance. Shortly after, Grimmer & Liu (2026) proved an information-theoretic lower bound in the high-dimensional setting for a broad class of LMO-based algorithms. For uniformly convex sets of order $p > 2$ (e.g., ℓ_p-balls), Kerdreux et al. (2021) established an upper bound of $\mathcal{O}(1/T^{p/(p-1)})$, but no matching lower bound was known. The goal was to prove lower bounds for the uniformly convex setting based on the techniques used by Halbey et al. (2026) or Grimmer & Liu (2026).

What the agent did and found. The agent began by studying both existing lower-bound techniques and attempting to generalize the high-dimensional construction by Grimmer & Liu (2026) to ℓ_p-balls. This did not succeed: the construction relies on decomposing strongly convex sets as intersections of shifted Euclidean balls, and the agent did not find a direct analogue for uniformly convex sets of order $p > 2$. Following Commandment IX, the agent documented this negative result and pivoted to the alternative approach of Halbey et al. (2026), which analyzes the FW iterates directly on a worst-case instance. The agent derived the FW dynamics on ℓ_p-balls in closed form and verified each component numerically (Commandment X). Experiments across multiple values of p revealed that the iterates alternate in sign and settle onto a low-dimensional curve whose shape can be characterized analytically, which suggested the right proof strategy. The main result establishes a lower bound of $\Omega(1/T^{p/(p-1)})$ for vanilla FW on p-uniformly convex sets for any $p \geq 3$, matching the upper bound of Kerdreux et al. (2021) and resolving the open question for this regime. The proof provides explicit convergence constants, all verified numerically to $< 0.2\%$ relative error (Figure 3). The case $p \in (2, 3)$ remains open: numerical evidence supports the same rate, but the proof technique does not extend.

Lessons. The correct proof strategy emerged from the agent's numerical exploration: patterns observed in the iterates suggested the right analytical approach, and the key constants were first estimated computationally before being derived in closed form. This "conjecture from computation, then prove" loop, enabled by the framework's emphasis on creating verification scripts alongside every mathematical claim (Commandment X), is a natural workflow for this type of problem. The failed generalization of Grimmer & Liu (2026) was equally informative: it helped us understand which parts of the proof are hard to extend to the uniformly convex setting, guiding the pivot to the successful approach. Following Commandment IX, this negative result was documented thoroughly rather than discarded.

4.2. Multi-Variable Dual Tightening for Mixed-Integer Optimization

Setting. Boscia (Hendrych et al., 2025) is a Frank-Wolfe-based branch-and-bound solver for mixed-integer nonlinear optimization over polytopes ($\min_{x \in X \cap \mathbb{Z}^J} f(x)$ with f smooth convex), where $X \subseteq \mathbb{R}^n$. A key pruning mechanism is *dual tightening*. At a relaxed solution x^t with gradient $g = \nabla f(x^t)$ and Frank-Wolfe dual gap $\gamma(x^t) = \max_{v \in X} \langle g, x^t - v \rangle$, convexity implies that any feasible point $x \in X$ with objective value at most some upper bound UB satisfies $g_j(x_j - \ell_j) \leq \text{RHS}$ for each variable j, where $\text{RHS} := \text{UB} - f(x^t) + \gamma(x^t)$ and ℓ_j is the lower bound of x_j. This allows variables to be fixed one at a time. The project investigated whether this extends to *subsets*: for a set S of variables at their lower bounds, $\sum_{j \in S} g_j(x_j - \ell_j) \leq \text{RHS}$, so when the combined gradient contribution exceeds the budget, a *conflict constraint* prevents all variables from simultaneously deviating from their current bounds.

What the agent did and found. The agent started from Boscia's existing single-variable dual tightening result, identified the natural generalization via the convexity inequality, and formulated and proved a multi-variable dual tightening theorem with corollaries for pairwise and higher-order binary conflicts. Before implementation, the agent first tried to verify the proof both symbolically, using `Symbolics.jl` with 2,387 checks, and numerically, using a script that exhaustively enumerated all 2^n feasible points for small instances (487 checks). This verification caught an error in the initial derivation: the bound for the at-least set constraint had been inverted, which would have led to overly aggressive fixings for upper-bound variables. The agent then implemented a `ConflictGraph` data structure with constraint propagation and integrated it into Boscia via two *callbacks* (Figure 4), requiring no source modifications beyond fixing a pre-existing `Dict` type bug. Across 33 instances in six problem categories, partition-constrained instances show the strongest improvement (up to 18.9% node reduction), while

4.1 Callback architecture

The conflict graph is integrated via two standard Boscia callbacks. No Boscia source modifications are required beyond the existing Dict type fix for `settings.tightening` (commit `c8f86437b`).

`propagate_bounds(tree, node)`
: Called at each node *before* the Frank-Wolfe solve. Propagates conflict-implied fixings from the root-derived conflict graph into `node.local.bounds`, rebuilds the LMO, and cleans the active set (see Section 4.2).

`bnb_callback(tree, node)`
: Called *after* each node is processed. At the root (`node.std.id` $= 1$): derives conflicts into the global graph and stores a gradient/iterate snapshot for re-scanning. At non-root nodes: checks whether the incumbent improved and, if so, re-scans the root snapshot with the tighter RHS $= \mathrm{UB}_{\mathrm{new}} - f(x^t_{\mathrm{root}}) + \tau \cdot \gamma_{\mathrm{root}}$.

Figure 4. A screenshot [*sic*] from the agent's report: the callback architecture for the Boscia case study. The conflict graph is integrated via `propagate_bounds` and `bnb_callback`, without modifying Boscia's source code.

all instances produce identical optimal objectives in both modes. As expected, separable quadratic instances show no benefit because diagonal objectives create no cross-variable coupling, confirming the theoretical prediction.

Lessons. This case study shows that the framework is effective for projects that combine theorem proving, verification, implementation, and experiments in a single workflow. The verification-first approach (Commandment X) was crucial for correctness: it caught the inverted at-least bound bug before it entered the experiment phase. The negative results were useful as well. The lack of improvement on separable instances matched the theory, while the $26\times$ overhead on a sparse regression instance with 150 indicator variables exposed a concrete bottleneck and pointed to straightforward fixes, including better data structures and a cap on propagated conflicts. Following Commandment IX, these outcomes were all documented in the report, which made the evaluation more transparent and more useful for guiding future improvements.

4.3. Weight Reconstruction in Large Language Model Pruning

Setting. Pruning large language models (LLMs) reduces memory and compute costs by zeroing out weights, i.e., selecting a binary sparsity mask $M \in \{0,1\}^{d_{\text{out}} \times d_{\text{in}}}$ per weight matrix (cf., e.g., Zimmer et al., 2023a; Frantar & Alistarh, 2023; Sun et al., 2024). The constraints on M determine the sparsity pattern and, with it, the potential for hardware acceleration: unstructured sparsity removes arbitrary individual weights, while semi-structured patterns such as $N{:}M$ impose structure that is more amenable to hardware acceleration. The core challenge across all settings is *mask selection*: choosing which weights to zero out so that the pruned network's output remains close to the original. Once a mask is fixed, the pruned model's performance degrades compared to the dense original; one way to counteract this is *weight reconstruction*, i.e., adjusting the surviving weights to compensate for the removed connections (Frantar & Alistarh, 2023).

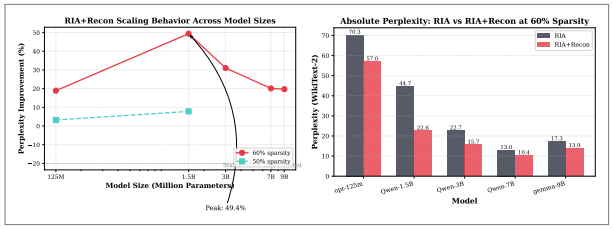

Figure 5. Plots [*sic*] from the agent's report: relative and absolute perplexity improvements from the weight-reconstruction method across tested model sizes.

The project started with a concrete task: we had developed a pruning approach that aimed to find better masks, but it produced inconsistent results, sometimes failing catastrophically. The agent was provided with an existing codebase containing implementations of several pruning methods and the LaTeX derivation of our approach, and instructed to analyze why it failed, fix or replace the method, and empirically beat a set of baselines (Sun et al., 2024; Zhang et al., 2023) at 60% sparsity.

What the agent did and found. The agent first established that the existing approach was mathematically flawed and could not be repaired. While analyzing *why* it failed, the agent studied how pruning distorts the post-layer activations of each weight matrix and observed a severe imbalance: some rows lose over 50% of their activation-weighted output magnitude while others lose less than 10%. This byproduct of debugging led the agent to propose a simple post-pruning weight correction that restores the activation balance across rows and columns. Following Commandment VIII, the agent first computed an oracle bound via least-squares reconstruction to determine the theoretical limit, then validated the new method through the tiered evaluation protocol (Commandment VII) across five model scales.

Results and lessons. The method consistently reduces perplexity by 18–50% across five model scales (125M to 9B parameters), three architectures (OPT, Qwen, Gemma), and two pruning methods (RIA, Wanda). It requires only 10 lines of code, adds less than 1% computational overhead, and needs no hyperparameter tuning. The oracle comparison shows that this simple heuristic captures 92% of the improvement achievable by full least-squares reconstruction, leaving little room for more sophisticated approaches. The original task was to fix a broken pruning mask; the actual outcome was a novel weight reconstruction method. The commandments forced the agent to analyze *why* the approach failed rather than simply trying the next idea, and this systematic analysis led to the discovery.

4.4. Column Ordering in LLM Quantization

Setting. Post-training quantization compresses a pretrained language model by representing its weights in lower precision, substantially reducing memory footprint and enabling deployment on consumer-grade hardware. GPTQ (Frantar et al., 2023), a widely used method, processes each weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ column by column to minimize the layer-wise reconstruction error $\|(W - \hat{W})X\|_F^2$, where $\hat{W}$ is the quantized matrix and X contains calibration activations. Each column's rounding error is propagated to subsequent columns via the inverse of the Hessian $H = 2XX^\top$. The order in which columns are processed therefore affects final model quality. A widely used post-publication variant, "act-order," sorts columns by descending Hessian diagonal, with the intuition that high-sensitivity columns benefit from having more subsequent columns available for error compensation. The agent was asked to investigate whether better orderings exist, how the effect depends on model architecture, and whether the findings survive full-model validation. Calibration data is drawn from C4 (Raffel et al., 2020); quality is measured by perplexity on WikiText (Merity et al., 2016).

What the agent did and found. The agent began with a mathematical analysis of why column ordering matters, then implemented and compared seven ordering strategies, first on single weight matrices and then at full model scale. Following Commandment X, it created verification scripts for all error-propagation and refinement formulas before running benchmarks (Figure 6). The first round of experiments treated a single weight matrix as a controlled setting: the agent varied the ordering while holding calibration data, bit width, and reconstruction procedure fixed. It then escalated to full-model evaluations only after these checks passed. Cross-architecture validation (Commandment VII) across Qwen, Llama, Gemma, Mistral, and Yi revealed the central finding: the ordering effect varies by more than two orders of magnitude across architectures. Column ordering reduces perplexity by 74% on Llama-3.1-8B but only 0.1% on Gemma-2-9B at 4-bit; on Qwen-1.5B alone the effect is 20%, which would have hidden this range. Alternatives that incorporate quantization-error magnitude sometimes outperform act-order, for example at 3-bit on certain architectures, but no strategy dominates consistently across all architectures and bit widths.

Results. The most important outcome was not a single new ordering rule, but a characterization of when ordering matters. On architectures where GPTQ is sensitive to column order, act-order remains a strong baseline, yet variants that combine Hessian sensitivity with predicted quantization error can improve further. On architectures where the ordering effect is negligible, additional ordering sophistication contributes little. Nine of the 24 experiments produced negative results, each documented with the same rigor as positive ones (Commandment IX); many approaches failed because GPTQ's ordinary-least-squares error propagation already minimizes the correlations those methods attempted to exploit. A critical implementation bug in group quantization was caught because the agent investigated a failure rather than abandoning the method (Commandment V): pre-computing scale parameters from initial instead of error-propagated weights produced catastrophic results (perplexity 437 versus 9.22 after the fix). The report documents all 24 experiments and 11 key findings.

Verification

What: $GPTQ_{new}$ error propagation and refinement formulas
Method: Numeric tests on small matrices (32×64, 32×128)
Script: `scripts/verify_gptq_new.py`
Outcome: All 5 tests pass. No bugs found. One known approximation documented (within-chunk propagation in $GPTQ_{new}$).

Status: Complete.

Figure 6. A screenshot [*sic*] from the agent's report: before benchmarking GPTQ variants, the agent audited the error-propagation and refinement formulas through numeric tests on small matrices (Commandment X).

Lessons. This case study shows why staged validation matters in ML experiments. A result that looks decisive on one model family can be misleading when the same intervention is nearly irrelevant elsewhere. The negative results were more informative than the positive ones: each failure clarified *why* simpler methods work, revealing that GPTQ's OLS-based error propagation already handles what more sophisticated alternatives attempt. With four GPUs, the agent ran independent model evaluations in parallel (one per GPU, Commandment C1), efficiently covering five model families with multiple configurations while keeping the experimental record in a single report.

5. Discussion and Conclusion

The case studies share a common pattern. The agent did not simply generate a final proof, implementation, or answer; it maintained a research state across many small steps: failed proof attempts, numerical evidence, symbolic checks, implementation choices, benchmark results, and open questions. This is where we believe agentic workflows are currently most useful. They are not reliable enough to be treated as independent sources of truth, but they are increasingly effective at producing the scaffolding around a result: executable checks, derivation logs, counterexample searches, reproduction scripts, and well-organized negative evidence.

Verification is therefore an operating mode, not a final cleanup step. In the GPTQ case, the agent audited error-propagation formulas before benchmarking; in the FW case, it checked closed-form recurrences against high-precision

numerical dynamics; in the Boscia case, symbolic checks and exhaustive enumeration caught an incorrect theorem variant before it reached the solver. These forms of verification differ from fully formal proof, but they make the agent's work inspectable and reduce the chance that a plausible story survives simply because no one tried to break it.

Limitations. Natural-language proofs and citations still demand careful manual inspection (Commandment III, Commandment X); subtle implementation errors can invalidate apparently strong computational evidence; and long sessions can exceed the model context, making persistent artifacts such as `report.tex`, `TODO.md`, logs, and commits essential. Long frontier-model runs also incur nontrivial cost, although much wall-clock time is often spent waiting for experiments rather than generating tokens. We release the instruction set, templates, and container definitions with the submission materials to make this workflow easier to inspect, reproduce, and adapt.

Impact Statement

This paper presents work whose goal is to advance AI-assisted mathematical and machine-learning research. A central risk of such workflows is that plausible but false mathematical claims, experimental results, or citations may be produced more quickly and with greater apparent authority. For this reason, the framework emphasizes human audit, independent verification, and persistent records of failures and uncertainties: the researcher remains responsible for judging correctness, novelty, and whether a result should be trusted.

References

Abouzaid, M., Blumberg, A. J., Hairer, M., Kileel, J., Kolda, T. G., Nelson, P. D., Spielman, D., Srivastava, N., Ward, R., Weinberger, S., and Williams, L. First Proof, February 2026.

Achim, T., Best, A., Bietti, A., Der, K., Fédérico, M., Gukov, S., Halpern-Leistner, D., Henningsgard, K., Kudryashov, Y., Meiburg, A., Michelsen, M., Patterson, R., Rodriguez, E., Scharff, L., Shanker, V., Sicca, V., Sowrirajan, H., Swope, A., Tamas, M., Tenev, V., Thomm, J., Williams, H., and Wu, L. Aristotle: IMO-level Automated Theorem Proving, October 2025. URL https://arxiv.org/abs/2510.01346v2.

Alexeev, B. and Mixon, D. G. Forbidden Sidon subsets of perfect difference sets, featuring a human-assisted proof, January 2026.

Alfarano, A., Charton, F., and Hayat, A. Global Lyapunov functions: A long-standing open problem in mathematics, with symbolic transformers, October 2024. URL http://arxiv.org/abs/2410.08304.

Anomaly. Opencode, February . URL https://github.com/anomalyco/opencode.

Anthropic. Claude code. URL https://code.claude.com/docs/en/overview.

Avigad, J. Mathematicians in the Age of AI. 2026.

AxiomMath. AxiomProver reports perfect score on Putnam 2025, 2025. URL https://github.com/AxiomMath/putnam2025. GitHub repository, accessed 2026-03-06.

Bragg, J., D'Arcy, M., Balepur, N., Bareket, D., Dalvi, B., Feldman, S., Haddad, D., Hwang, J. D., Jansen, P., Kishore, V., Majumder, B. P., Naik, A., Rahamimov, S., Richardson, K., Singh, A., Surana, H., Tiktinsky, A., Vasu, R., Wiener, G., Anastasiades, C., Candra, S., Dunkelberger, J., Emery, D., Evans, R., Hamada, M., Huff, R., Kinney, R., Latzke, M., Lochner, J., Lozano-Aguilera, R., Nguyen, C., Rao, S., Tanaka, A., Vlahos, B., Clark, P., Downey, D., Goldberg, Y., Sabharwal, A., and Weld, D. S. AstaBench: Rigorous Benchmarking of AI Agents with a Scientific Research Suite, October 2025.

Breen, B., Tredici, M. D., McCarran, J., Mijares, J. A., Yin, W. W., Sulimany, K., Taylor, J. M., Koppens, F. H. L., and Englund, D. Ax-Prover: A Deep Reasoning Agentic Framework for Theorem Proving in Mathematics and Quantum Physics, November 2025.

Bubeck, S., Coester, C., Eldan, R., Gowers, T., Lee, Y. T., Lupsasca, A., Sawhney, M., Scherrer, R., Sellke, M., Spears, B. K., Unutmaz, D., Weil, K., Yin, S., and Zhivotovskiy, N. Early science acceleration experiments with GPT-5, November 2025.

Carbone, L. Advancing mathematics research with generative AI, December 2025. URL http://arxiv.org/abs/2511.07420.

Cemri, M., Agrawal, S., Gupta, A., Liu, S., Cheng, A., Mang, Q., Naren, A., Erdogan, L. E., Sen, K., Zaharia, M., Dimakis, A., and Stoica, I. AdaEvolve: Adaptive LLM Driven Zeroth-Order Optimization, February 2026.

Chen, Z., Chen, S., Ning, Y., Zhang, Q., Wang, B., Yu, B., Li, Y., Liao, Z., Wei, C., Lu, Z., Dey, V., Xue, M., Baker, F. N., Burns, B., Adu-Ampratwum, D., Huang, X., Ning, X., Gao, S., Su, Y., and Sun, H. ScienceAgentBench: Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery, March 2025. URL http://arxiv.org/abs/2410.05080.

Chervonyi, Y., Trinh, T. H., Olšák, M., Yang, X., Nguyen, H., Menegali, M., Jung, J., Kim, J., Verma, V., Le, Q. V., and Luong, T. Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2, December 2025. URL http://arxiv.org/abs/2502.03544.

Chuharski, J., Collins, E. R., and Meringolo, M. Mining math conjectures from LLMs: A pruning approach. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=aYlKvzY6ob.

Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., Lackenby, M., Williamson, G., Hassabis, D., and Kohli, P. Advancing mathematics by guiding human intuition with AI. *Nature*, 600(7887): 70–74, December 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-04086-x. URL https://www.nature.com/articles/s41586-021-04086-x.

de Moura, L. 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.

Dekoninck, J., Petrov, I., Minchev, K., Balunovic, M., Vechev, M., Marinov, M., Drencheva, M., Konova, L., Shumanov, M., Tsvetkov, K., Drenchev, N., Todorov, L., Nikolova, K., Georgiev, N., Kalinkova, V., and Ismoldayev, M. The Open Proof Corpus: A Large-Scale Study of LLM-Generated Mathematical Proofs, January 2026.

Diez, C.-P., da Maia, L., and Nourdin, I. Mathematical research with GPT-5: A Malliavin-Stein experiment, September 2025.

Dobriban, E. Solving a Research Problem in Mathematical Statistics with AI Assistance, December 2025. URL http://arxiv.org/abs/2511.18828.

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R. Ruiz, F. J., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., and Kohli, P. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53, October 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-05172-4. URL https://www.nature.com/articles/s41586-022-05172-4.

Feldman, M. and Karbasi, A. Gödel Test: Can Large Language Models Solve Easy Conjectures?, September 2025.

Feng, T. Eigenweights for arithmetic Hirzebruch Proportionality, February 2026.

Feng, T., Jung, J., Kim, S.-h., Pagano, C., Gukov, S., Tsai, C.-C., Woodruff, D., Javanmard, A., Mokhtari, A., Hwang, D., Chervonyi, Y., Lee, J. N., Bingham, G., Trinh, T. H., Mirrokni, V., Le, Q. V., and Luong, T. Aletheia tackles FirstProof autonomously, February 2026a.

Feng, T., Trinh, T. H., Bingham, G., Hwang, D., Chervonyi, Y., Jung, J., Lee, J., Pagano, C., Kim, S.-h., Pasqualotto, F., Gukov, S., Lee, J. N., Kim, J., Hou, K., Ghiasi, G., Tay, Y., Li, Y., Kuang, C., Liu, Y., Lin, H., Liu, E. Z., Nayakanti, N., Yang, X., Cheng, H.-T., Hassabis, D., Kavukcuoglu, K., Le, Q. V., and Luong, T. Towards Autonomous Mathematics Research, February 2026b. URL http://arxiv.org/abs/2602.10177.

Ferrag, M. A., Tihanyi, N., and Debbah, M. From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review, April 2025.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, March 2023. URL http://arxiv.org/abs/2210.17323.

George, R. J., Huang, S., Song, P., and Anandkumar, A. LeanProgress: Guiding Search for Neural Theorem Proving via Proof Progress Prediction, January 2026.

Georgiev, B., Gómez-Serrano, J., Tao, T., and Wagner, A. Z. Mathematical exploration and discovery at scale, November 2025. URL https://arxiv.org/abs/2511.02864v3.

Glazer, E., Erdil, E., Besiroglu, T., Chicharro, D., Chen, E., Gunning, A., Olsson, C. F., Denain, J.-S., Ho, A., Santos, E. d. O., Järviniemi, O., Barnett, M., Sandler, R., Vrzala, M., Sevilla, J., Ren, Q., Pratt, E., Levine, L., Barkley, G., Stewart, N., Grechuk, B., Grechuk, T., Enugandla, S. V., and Wildon, M. FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI, December 2025.

Google. Build, debug & deploy with AI. URL https://geminicli.com/.

Grimmer, B. and Liu, N. Lower bounds for linear minimization oracle methods optimizing over strongly convex sets. *arXiv preprint arXiv:2602.22608*, 2026.

Haase, J. and Pokutta, S. Human-AI Co-Creativity: Exploring Synergies Across Levels of Creative Collaboration. pp. 205–221. 2026. doi: 10.1016/B978-0-443-34073-4.00009-5. URL http://arxiv.org/abs/2411.12527.

Halbey, J., Deza, D., Zimmer, M., Roux, C., Stellato, B., and Pokutta, S. Lower bounds for frank-wolfe on strongly convex sets. *arXiv preprint arXiv:2602.04378*, 2026.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Handa, K., Stern, M., Huang, S., Hong, J., Durmus, E., McCain, M., Yun, G., Alt, AJ., Millar, T., Tamkin, A., Leibrock, J., Ritchie, S., and Ganguli, D. Introducing anthropic interviewer: What 1,250 professionals told us about working with AI. https://anthropic.com/research/anthropic-interviewer, 2025-12-04, 2025.

Hendrych, D., Troppens, H., Besançon, M., and Pokutta, S. Convex mixed-integer optimization with Frank–Wolfe methods. *Mathematical Programming Computation*, 17 (4):731–757, December 2025. ISSN 1867-2957. doi: 10.1007/s12532-025-00288-w. URL https://doi.org/10.1007/s12532-025-00288-w.

Henkel, J. The Mathematician's Assistant: Integrating AI into Research Practice, August 2025. URL http://arxiv.org/abs/2508.20236.

Huang, Y. and Yang, L. F. Winning Gold at IMO 2025 with a Model-Agnostic Verification-and-Refinement Pipeline, September 2025.

Huang, Y., Lin, X., Liu, Z., Cao, Q., Xin, H., Wang, H., Li, Z., Song, L., and Liang, X. MUSTARD: Mastering Uniform Synthesis of Theorem and Proof Data, May 2024. URL http://arxiv.org/abs/2402.08957.

Hubert, T., Mehta, R., Sartran, L., Horváth, M. Z., Žužić, G., Wieser, E., Huang, A., Schrittwieser, J., Schroecker, Y., Masoom, H., Bertolli, O., Zahavy, T., Mandhane, A., Yung, J., Beloshapka, I., Ibarz, B., Veeriah, V., Yu, L., Nash, O., Lezeau, P., Mercuri, S., Sönne, C., Mehta, B., Davies, A., Zheng, D., Pedregosa, F., Li, Y., von Glehn, I., Rowland, M., Albanie, S., Velingker, A., Schmitt, S., Lockhart, E., Hughes, E., Michalewski, H., Sonnerat, N., Hassabis, D., Kohli, P., and Silver, D. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pp. 1–3, November 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09833-y. URL https://www.nature.com/articles/s41586-025-09833-y.

Ivanisvili, P. and Xie, X. Counterexample to majority optimality in NICD with erasures, October 2025.

Jansen, P., Tafjord, O., Radensky, M., Siangliulue, P., Hope, T., Mishra, B. D., Majumder, B. P., Weld, D. S., and Clark, P. CodeScientist: End-to-End Semi-Automated Scientific Discovery with Code-based Experimentation, March 2025.

Jiang, J., Ding, T., and Zhu, Z. DeltaEvolve: Accelerating Scientific Discovery through Momentum-Driven Evolution, February 2026.

Jordan, K., Jin, Y., Boza, V., Jiacheng, Y., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6(3):4, 2024.

Ju, H. and Dong, B. AI for Mathematics: Progress, Challenges, and Prospects, February 2026. URL http://arxiv.org/abs/2601.13209.

Karpathy, A. autoresearch, 2026. URL https://github.com/karpathy/autoresearch. GitHub repository, accessed 2026-03-08.

Kera, H., Pelleriti, N., Ishihara, Y., Zimmer, M., and Pokutta, S. Computational Algebra with Attention: Transformer Oracles for Border Basis Algorithms, May 2025. URL https://arxiv.org/abs/2505.23696.

Kerdreux, T., d'Aspremont, A., and Pokutta, S. Projection-free optimization on uniformly convex sets. In *International conference on artificial intelligence and statistics*, pp. 19–27. PMLR, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Knuth, D. E. Claude's cycles, 2026. URL https://www-cs-faculty.stanford.edu/~knuth/papers/claude-cycles.pdf.

Kumarappan, A., Tiwari, M., Song, P., George, R. J., Xiao, C., and Anandkumar, A. LeanAgent: Lifelong Learning for Formal Theorem Proving, March 2025.

Lange, R. T., Imajuku, Y., and Cetin, E. ShinkaEvolve: Towards Open-Ended And Sample-Efficient Program Evolution, September 2025.

Lasby, M., Zimmer, M., Pokutta, S., and Schultheis, E. Compressed sparse tiles for memory-efficient unstructured and semi-structured sparsity. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025. URL https://openreview.net/forum?id=iso0KV2HVq.

Lee, J. and Seo, J. Lower bounds for multivariate independence polynomials and their generalisations, February 2026.

Li, Z., Liu, L., Liang, C., Chen, W., and Zhao, T. Normuon: Making muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., and Jin, C. Goedel-Prover-V2: Scaling Formal Theorem Proving with Scaffolded Data Synthesis and Self-Correction, August 2025.

Lindberg, J., Zachariah, A., Boston, N., and Lesieutre, B. C. The Distribution of the Number of Real Solutions to the Power Flow Equations, October 2020.

Liu, Y., Wang, B., Li, P., and Liu, Y. AI Mathematician as a Partner in Advancing Mathematical Discovery – A Case Study in Homogenization Theory, October 2025. URL http://arxiv.org/abs/2510.26380.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lu, C., Lu, C., Lange, R. T., Foerster, J., Clune, J., and Ha, D. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, September 2024. URL http://arxiv.org/abs/2408.06292.

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., Köppe, T., Millikin, K., Gaffney, S., Elster, S., Broshear, J., Gamble, C., Milan, K., Tung, R., Hwang, M., Cemgil, T., Barekatain, M., Li, Y., Mandhane, A., Hubert, T., Schrittwieser, J., Hassabis,

D., Kohli, P., Riedmiller, M., Vinyals, O., and Silver, D. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, June 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06004-9. URL https://www.nature.com/articles/s41586-023-06004-9.

mathlib Community, T. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381, January 2020. doi: 10.1145/3372885.3373824.

McCain, M., Millar, T., Huang, S., Eaton, J., Handa, K., Stern, M., Tamkin, A., Kearney, M., Durmus, E., Shen, J., Hong, J., Calvert, B., Chan, J. S., Mosconi, F., Saunders, D., Neylon, T., Nicholas, G., Pollack, S., Clark, J., and Ganguli, D. Measuring AI agent autonomy in practice. https://anthropic.com/research/measuring-agent-autonomy, 2026.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mishra, A., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Mishra, C., Moulik, S. R., and Sarkar, R. Mathematical conjecture generation using machine intelligence, June 2023. URL http://arxiv.org/abs/2306.07277.

Mundinger, K., Pokutta, S., Spiegel, C., and Zimmer, M. Extending the continuum of six-colorings. *Geombinatorics Quarterly*, XXXIV, 2024. URL https://geombina.uccs.edu/past-issues/volume-xxxiv.

Mundinger, K., Zimmer, M., Kiem, A., Spiegel, C., and Pokutta, S. Neural discovery in mathematics: Do machines dream of colored planes? In *Forty-Second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=7Tp9zjP9At.

Noorani, S., Kiyani, S., Pappas, G., and Hassani, H. Human-AI Collaborative Uncertainty Quantification, October 2025. URL http://arxiv.org/abs/2510.23476.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. AlphaEvolve: A coding agent for scientific and algorithmic discovery, June 2025. URL https://arxiv.org/abs/2506.13131v1.

OpenAI. Codex | AI Coding Partner from OpenAI. URL https://openai.com/codex/.

Ospanov, A., Farnia, F., and Yousefzadeh, R. APOLLO: Automated LLM and Lean Collaboration for Advanced Formal Reasoning, November 2025.

Pelleriti, N., Spiegel, C., Liu, S., Martínez-Rubio, D., Zimmer, M., and Pokutta, S. Neural Sum-of-Squares: Certifying the Nonnegativity of Polynomials with Transformers, October 2025. URL https://arxiv.org/abs/2510.13444.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Ren, Y., Li, J., and Qi, Y. MerLean: An Agentic Framework for Autoformalization in Quantum Computation, February 2026.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition, July 2025.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Roux, C., Zimmer, M., d'Aspremont, A., and Pokutta, S. Don't be greedy, just relax! pruning llms via frank-wolfe. *arXiv preprint arXiv:2510.13713*, 2025.

Ruan, K., Xu, Y., Gao, Z.-F., Liu, Y., Guo, Y., Wen, J.-R., and Sun, H. Discovering physical laws with parallel symbolic enumeration. *Nature Computational Science*, 6 (1):53–66, January 2026. ISSN 2662-8457. doi: 10.1038/s43588-025-00904-8. URL https://www.nature.com/articles/s43588-025-00904-8.

Salim, A. Accelerating mathematical research with language models: A case study of an interaction with GPT-5-Pro on a convex analysis problem, October 2025.

Schmidgall, S. and Moor, M. AgentRxiv: Towards Collaborative Autonomous Research, March 2025.

Schmidgall, S., Su, Y., Wang, Z., Sun, X., Wu, J., Yu, X., Liu, J., Moor, M., Liu, Z., and Barsoum, E. Agent Laboratory: Using LLM Agents as Research Assistants, June 2025.

Schmitt, J. Extremal descendant integrals on moduli spaces of curves: An inequality discovered and proved in collaboration with AI, December 2025.

Semenov, A., Pagliardini, M., and Jaggi, M. Benchmarking optimizers for large language model pretraining. *arXiv preprint arXiv:2509.01440*, 2025.

Sharma, A. OpenEvolve: An open-source evolutionary coding agent. GitHub, 2025.

Si, C., Zhang, D., and Shen, W. Adamuon: Adaptive muon optimizer. *arXiv preprint arXiv:2507.11005*, 2025.

Song, P., Yang, K., and Anandkumar, A. Lean Copilot: Large Language Models as Copilots for Theorem Proving in Lean, May 2025. URL http://arxiv.org/abs/2404.12534.

Sothanaphan, N. Resolution of Erdős Problem #728: A writeup of Aristotle's Lean proof, January 2026.

Su, W. J. You Are the Best Reviewer of Your Own Papers: An Owner-Assisted Scoring Mechanism, June 2022.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A Simple and Effective Pruning Approach for Large Language Models, May 2024. URL http://arxiv.org/abs/2306.11695.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, January 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06747-5. URL https://www.nature.com/articles/s41586-023-06747-5.

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. PutnamBench: Evaluating Neural Theorem-Provers on the Putnam Mathematical Competition, November 2024.

Udrescu, S.-M. and Tegmark, M. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, April 2020. doi: 10.1126/sciadv.aay2631. URL https://www.science.org/doi/10.1126/sciadv.aay2631.

Varambally, S., Voice, T., Sun, Y., Chen, Z., Yu, R., and Ye, K. Hilbert: Recursively Building Formal Proofs with Informal Reasoning, September 2025. URL http://arxiv.org/abs/2509.22819.

Wang, H., Xin, H., Zheng, C., Li, L., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., Liao, H., and Liang, X. LEGO-Prover: Neural Theorem Proving with Growing Libraries, October 2023. URL http://arxiv.org/abs/2310.00656.

Wang, H., Xie, R., Wang, Y., Gao, G., Yu, X., and Dong, B. Aria: An Agent For Retrieval and Iterative Auto-Formalization via Dependency Graph, October 2025a. URL http://arxiv.org/abs/2510.04520.

Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. TheoremLlama: Transforming General-Purpose LLMs into Lean4 Experts, October 2024. URL http://arxiv.org/abs/2407.03203.

Wang, Y., Bennani, M., Martens, J., Racanière, S., Blackwell, S., Matthews, A., Nikolov, S., Cao-Labora, G., Park, D. S., Arjovsky, M., Worrall, D., Qin, C., Alet, F., Kozlovskii, B., Tomašev, N., Davies, A., Kohli, P., Buckmaster, T., Georgiev, B., Gómez-Serrano, J., Jiang, R., and Lai, C.-Y. Discovery of Unstable Singularities, September 2025b. URL http://arxiv.org/abs/2509.14185.

Wang, Y., Su, S.-R., Zeng, Z., Xu, E., Ren, L., Yang, X., Huang, Z., He, X., Ma, L., Peng, B., Cheng, H., He, P., Chen, W., Wang, S., Du, S. S., and Shen, Y. ThetaEvolve: Test-time Learning on Open Problems, November 2025c.

Wang, Z., Yang, B., Li, C., Zhang, Y., Zhou, S., Dong, B., and Wen, Z. Translating Informal Proofs into Formal Proofs Using a Chain of States, December 2025d. URL http://arxiv.org/abs/2512.10317.

Weng, K., Du, L., Li, S., Lu, W., Sun, H., Liu, H., and Zhang, T. Autoformalization in the Era of Large Language Models: A Survey, May 2025. URL http://arxiv.org/abs/2505.23486.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with Large Language Models, May 2022. URL http://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, May 2024. URL http://arxiv.org/abs/2405.14333.

Xuejun, Y., Zhong, J., Feng, Z., Zhai, P., Yousefzadeh, R., Ng, W. C., Liu, H., Shou, Z., Xiong, J., Zhou, Y., Ong, C. B., Sugiarto, A. J., Zhang, Y., Tai, W. M., Cao, H., Lu, D., Sun, J., Xu, Q., Xin, S., and Li, Z. Mathesis: Towards Formal Theorem Proving from Natural Languages, June 2025. URL http://arxiv.org/abs/2506.07047.

Yamada, Y., Lange, R. T., Lu, C., Hu, S., Lu, C., Foerster, J., Clune, J., and Ha, D. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search, April 2025. URL http://arxiv.org/abs/2504.08066.

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, October 2023. URL http://arxiv.org/abs/2306.15626.

Zhang, J., Petrui, C., Nikolić, K., and Tramèr, F. RealMath: A Continuous Benchmark for Evaluating Language Models on Research-Level Mathematics, October 2025a.

Zhang, L., Valentino, M., and Freitas, A. Autoformalization in the Wild: Assessing LLMs on Real-World Mathematical Definitions. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1720–1738, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.90. URL https://aclanthology.org/2025.emnlp-main.90/.

Zhang, R., Zhao, Y., Liu, Z., Wang, Z., and Zhang, Z. Muon+: Towards better muon via one additional normalization step. *arXiv preprint arXiv:2602.21545*, 2026.

Zhang, Y., Bai, H., Lin, H., Zhao, J., Hou, L., and Cannistraci, C. V. Plug-and-Play: An Efficient Post-training Pruning Method for Large Language Models. In *The Twelfth International Conference on Learning Representations*, October 2023. URL https://openreview.net/forum?id=Tr0lPx9woF.

Zheng, K., Han, J. M., and Polu, S. MiniF2F: A cross-system benchmark for formal Olympiad-level mathematics, February 2022.

Zheng, T., Deng, Z., Tsang, H. T., Wang, W., Bai, J., Wang, Z., and Song, Y. From automation to autonomy: A survey on large language models in scientific discovery. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17733–17750, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.895.

Zheng, T., Deng, Z., Tsang, H. T., Wang, W., Bai, J., Wang, Z., and Song, Y. From Automation to Autonomy: A Survey on Large Language Models in Scientific Discovery, September 2025b. URL http://arxiv.org/abs/2505.13259.

Zhou, Z., Cao, C., Feng, X., Li, X., Li, Z., Lu, X., Yao, J., Huang, W., Cheng, T., Zhang, J., Jiang, T., Xu, L., Zheng, Y., Miranda, B., Liu, T., Koyejo, S., Sugiyama, M., and Han, B. AlphaApollo: A system for deep agentic reasoning. *arXiv preprint arXiv:2510.06261*, 2025.

Zimmer, M., Andoni, M., Spiegel, C., and Pokutta, S. Perp: Rethinking the prune-retrain paradigm in the era of llms. *arXiv preprint arXiv:2312.15230*, 2023a.

Zimmer, M., Spiegel, C., and Pokutta, S. How I Learned To Stop Worrying And Love Retraining. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=_nF5imFKQI.

Zimmer, M., Spiegel, C., and Pokutta, S. Sparse model soups: A recipe for improved pruning via model averaging. In *The Twelfth International Conference on Learning Representations*, 2024.

Zimmer, M., Spiegel, C., and Pokutta, S. *Compression-aware Training of Neural Networks using Frank-Wolfe*, pp. 137–168. De Gruyter, Berlin, Boston, 2025. ISBN 9783111376776. doi: doi:10.1515/9783111376776-010. URL https://doi.org/10.1515/9783111376776-010.

Zimmer, M., Roux, C., Wagner, M., Hendrych, D., and Pokutta, S. SparseSwaps: Tractable LLM Pruning Mask Refinement at Scale, February 2026. URL http://arxiv.org/abs/2512.10922.

Zou, H., Feng, J., Zhao, H., and Shi, Y. Analytical Lyapunov Function Discovery: An RL-based Generative Approach, June 2025. URL http://arxiv.org/abs/2502.02014.

The appendix expands every section of the main paper to the level of detail of the long-form version. It includes the full five-level taxonomy with per-level descriptions, the full framework overview and infrastructure description, the Ten Commandments with their motivating failure modes, the domain-specific modules, full versions of the case studies, an extended discussion with limitations and future directions, and the related work.

A. Five-Level Taxonomy of AI Integration in Mathematical and ML Research

Inspired by Haase & Pokutta (2026), we propose a taxonomy of five levels that characterize how deeply AI is integrated into the research process, ranging from no AI involvement to fully autonomous research loops. These levels are not mutually exclusive, and a researcher might use different levels for different tasks, all within the same project. In particular, even (fully) autonomous systems can delegate subtasks to less autonomous components. This regularly happens also in our setup when subagents are spawned to accomplish subtasks. In general, the key lies in recognizing which level is appropriate for which task. Table 2 summarizes the taxonomy, and we describe each level in detail below.

Table 2. Five levels of AI integration in mathematical research. Each (not necessarily mutually exclusive) level represents a qualitatively different trade-off between agent autonomy and human involvement.

Level	Name	Tools	AI Tasks	Human Role
0	Classical	LaTeX, math. software	No AI integration	Everything
1	Consultant	LLM chatbots	Targeted queries for explanation, literature, brainstorming	Asks, evaluates
2	Typist	Editor plugins (Copilot, Cursor)	Code and text generation without execution	Thinks, reviews, decides
3	Collaborator	CLI coding agents	Human describes task, AI implements and iterates	Reviews each output, assigns next task
4	Research Assoc.	Our framework	Autonomous experiment loop following structured research plan	Steers, audits

Level 0: Classical. The classical level is the baseline of our taxonomy and the traditional mode of mathematical and ML research. The researcher uses all traditional computational tools, including typesetting software (e.g., LaTeX), mathematical software (e.g., Mathematica, MATLAB), and programming languages for custom implementations (e.g., Python, Julia, PyTorch), but no AI assistance. This remains the predominant mode of research and is perfectly appropriate. The goal of this paper is not to argue that AI should render it obsolete, but to show when and how AI can complement it.

Level 1: AI as Consultant. The researcher uses LLM-based chatbots (e.g., ChatGPT, Claude, Gemini) for specific queries and assistance. Typical cases include concept explanation (*Explain the difference between strong and weak duality in linear programming*), literature search (*What are the current best convergence rates for SGD with heavy-tailed noise?*), brainstorming (*What techniques exist for proving convergence of iterative algorithms when the operator is only approximately contractive?*), and debugging ideas (*Here is my proof attempt. Where does the argument break down?*). The core intellectual work remains with the researcher; the AI provides targeted assistance. The key skill is asking the right questions and crafting sufficiently detailed prompts to guide the AI toward a useful answer. A clear limitation is that the interaction is stateless across sessions unless the user manually provides context. *Getting started:* a web browser and access to an LLM chatbot (free tiers available from most providers). No setup required.

Level 2: AI as Typist. The researcher uses AI for code and text generation, ranging from tab completion (e.g., GitHub Copilot predicting the next line) to more complex prompt-based generation that produces entire functions or LaTeX paragraphs from a natural-language description. Every output is reviewed by the researcher and accepted, edited, or rejected. The defining characteristic of this level is that the AI generates code or text but neither executes nor iterates on the results. The researcher remains responsible for all design decisions, and the AI accelerates the writing process without closing the loop between implementation and evaluation. *Getting started:* install a code editor plugin (e.g., Cursor, or VS Code with GitHub Copilot).

Level 3: AI as Collaborator. The full implementation and execution are delegated to a *CLI coding agent*, i.e., a terminal-based tool (e.g., Claude Code (Anthropic), OpenCode (Anomaly), Codex CLI (OpenAI)) that can read and edit files, execute

shell commands, and iterate on results within a persistent project context. This differs qualitatively from Levels 1–2 because the agent possesses a much broader set of capabilities, including file modifications, code execution, and iteration based on results it has obtained, all within a single conversation. For a prompt like *"Implement the Frank-Wolfe algorithm for the semidefinite relaxation of max-cut, with step size* $\gamma_t = 2/(t+2)$*"* or *"Implement a learning rate scheduler with linear warmup,"* the agent reads the codebase, implements the algorithm, runs it, and re-evaluates if convergence shows unexpected behavior. The researcher describes each task in natural language and provides the necessary context, such as an existing codebase. After each completed task, the researcher reviews the output, decides *what* to do next, and assigns the next task; the agent handles *how*. At no point does the agent independently set the research direction. *Getting started:* install a CLI coding agent and start a session in the project directory.

Level 4: AI as Research Associate. The highest degree of autonomy in our taxonomy. The researcher arrives with a research idea (initial intuitions, failed strategies, partial results, or simply a well-posed question) and outlines a research plan: goals, metrics, constraints, approaches already tried, and promising directions to explore. The agent then formulates a detailed plan and autonomously executes an experiment loop: formalizing mathematical ideas, implementing approaches, running evaluations, recording results, analyzing outcomes, and updating both a structured research report and a `TODO.md`. It iterates this loop, continuously refining and expanding the plan, operating for hours to days to achieve the research goal or uncover something unexpected. To operate for extended periods, structured and clear instructions that govern scientific rigor, documentation, and verification are needed: our framework (Section 3) provides exactly these. The key difference from Level 3 is that the agent does not wait for human input between experiments but follows a research plan and a set of commandments encoding the norms of good scientific practice. Intermittent human review and course correction are an integral part of Level 4, not a fallback to Level 3: the researcher periodically inspects the report, adjusts priorities, and refines the research plan while the agent continues to execute autonomously. The researcher's role shifts from execution to direction-setting, periodic review, and evaluation. Level 4 is most appropriate when the search space is large.

Despite the guardrails described in Section 3, limitations remain. The agent may pursue an unproductive direction for too long, especially when the research plan lacks sufficient detail. Verification is only partially solved: while we provide strategies for symbolic and numerical verification of mathematical claims and implementations, a high (to full) degree of certainty requires the researcher to perform a rigorous review of the work. We consider this a feature, not a bug. Similarly, while the agent is instructed to search the literature, it cannot guarantee that its ideas are genuinely novel. Thorough knowledge of the related work remains the researcher's responsibility. As such, the researcher still faces a non-trivial amount of work both throughout and toward the end of a project: reviewing intermediate results and providing steering, verifying correctness, deciding what results merit publication, and confirming originality as well as adding context and interpretation. However, instead of conducting the entire research process alone, the researcher now externalizes parts of the work to a capable *research associate* who delivers a structured, well-documented report. Through repeated interactions of this kind, new results emerge in a process of Human-AI co-creation. *Getting started:* clone the project repository and follow the setup instructions in the `README.md`. The setup takes a couple of minutes, and the first autonomous experiments can begin immediately.

B. Framework: Long-Form Description

Overview and starting a new project. To start a new project, the researcher provides three things (Figure 1): a *research question* (problem formulation, hypotheses, evaluation criteria), the *tools, methods, and data* needed to investigate it (software stack, packages, datasets, compute resources), and any *prior work or domain knowledge* that should inform the investigation (existing codebase, LaTeX notes with derivations, references, preliminary results). In the following, we use the term *experiment* to refer to one (broad) agentic iteration loop with the researcher: depending on the context, this can be one proof attempt, an actual computational experiment, or the design of a new algorithm. The framework is built around CLI coding agents, e.g., Claude Code (Anthropic), Codex CLI (OpenAI), Gemini CLI (Google), or OpenCode (Anomaly), which operate inside a sandboxed container that provides a secure, isolated workspace. The typical workflow is: (1) the researcher begins in a project directory that contains the practical-layer materials described above (Figure 1). From this directory, they launch the sandbox and provide the research instructions to the agent. The more detailed the instructions, the better; we found it especially useful to provide a working codebase if one exists, along with a LaTeX write-up of the research problem and previously tried approaches. (2) The agent asks clarifying questions about scope, constraints, and evaluation metrics. (3) After this back-and-forth, the agent explores all relevant files and writes the final project-specific instructions into a persistent instruction file (`INSTRUCTIONS.md`), alongside the universal commandments that are already in place

(Appendix C). (4) The agent creates a plan and initializes `report.tex` and `TODO.md`, the two main artifacts of the research process. Upon approval by the researcher (or after further refinement of the plan), the agent begins autonomous execution and only requires human intervention in case of unexpected behavior or when the research plan needs adjustment.

Why CLI agents. Across our research workflows, three practical requirements arose repeatedly. CLI agents are *easy to use*: they fit naturally into local working environments, can be launched inside an existing project, and operate directly on local files without additional infrastructure. They remain fully *interactive*: the researcher can intervene at any point to inspect progress, redirect the investigation, stop execution, or restart with revised instructions. Finally, they are *extensible*: the toolchain can be readily extended with custom utilities; in our case, this included scripts for handling literature and LaTeX sources, extracting relevant algorithmic sections, and running specialized search and verification routines. The same mechanism also supports hard *guardrails*: automated checks can be triggered after file edits or experiment runs, enforcing formatting, running tests, or updating reports. Because CLI agents are maintained by model providers and evolve with model capabilities, while our rules sit on top, the framework automatically benefits from improvements to the underlying tools.

Infrastructure. Because the framework is built around CLI agents, the surrounding infrastructure can remain intentionally minimal. The sandbox confines all actions to a container, enabling unattended sessions without the risk of damaging the host system. For compute-intensive projects, a multi-node launcher dispatches independent experiments to remote Slurm nodes. We recommend using reproducible, project-local package managers (`uv` for Python, Julia's `Pkg`, among others).

Structured reporting and experiment tracking. All experimental progress is recorded in a single LaTeX file (`report.tex`) that accumulates experiments, derivations, and analysis, complemented by a `TODO.md` checklist for open questions, unverified claims, and deferred work. Each experiment subsection must contain the fields shown in the main paper, enforced by the commandments. Rather than introducing a separate experiment-tracking system, we use Git directly. Each experiment is recorded as a commit with a structured message of the form `exp(EXXX): <description> --<metric>=<value>`. Branches group related experiments, tags mark important outcomes, and Git's worktree feature allows multiple agent sessions to run concurrently on separate copies of the codebase without interference. This keeps the full experimental history lightweight, portable, and directly searchable through Git logs. Once running, each experiment follows the eight-step loop shown in Figure 2: *Explore* → *Plan* → *Implement* → *Evaluate* → *Analyze* → *Record* → *Commit* → *Iterate*. At the beginning of every session (or after a context window reset), the agent re-reads `report.tex`, `TODO.md`, and the `git log` to restore continuity.

C. The Ten Commandments (Full Text) and Domain Modules

At the core of our framework are the lessons we distilled through experimentation into ten commandments that apply independently of the specific domain and research problem. They form a major part of the instructions given to the agent. In deriving the ten commandments through continuous improvement of the agent's behavior, we followed three guiding principles: (1) *explicit over implicit:* language models follow instructions literally; implicit expectations ("obviously you should record your results") are reliably violated, so every important behavior must be stated as a rule; (2) *falsifiable over aspirational:* "be rigorous" is not a commandment, "change exactly one variable per experiment" is, allowing both human and agent to assess compliance; (3) *failure-driven over theory-driven:* every commandment exists because we observed a specific failure mode in practice, not because it seemed theoretically desirable.

The commandments are grouped into categories, each addressing a specific aspect of the research process. Below, we state each rule and describe the failure mode it addresses. We present slightly shortened versions for brevity; the full prompts are available in the repository. At the implementation level, each commandment is a prompt-engineering directive; we found that naming and structuring these behaviors as explicit rules makes them significantly easier to maintain, debug, and iterate on.

Integrity and Trust

The following three commandments address the integrity of the agent's promises and announced actions.

I. Never Break a Promise

If you say "I will do X," do it. Under-promise, over-deliver.

Failure mode: In early experiments, the agent frequently stated intentions ("I will now run the full evaluation") and then skipped steps or moved on to different tasks. After adding the commandment, the agent either follows through on all stated tasks or states upfront which tasks will be deferred and why.

II. Never Manipulate Evaluation

Do not change metrics, test sets, fixed hyperparameters, or problem definitions. Do not hardcode results or cherry-pick seeds.

Failure mode: The agent subtly changes evaluation conditions to make results look better. The LLM may adjust evaluation parameters "helpfully" to reach its goal, but this is not a genuine improvement. For instance, the agent changed the number of evaluation samples to "speed up evaluation", which happened to produce better metrics and created an unfair advantage over baseline methods.

III. Never Fabricate Citations

Every bibliography entry must be verified against the actual source before adding it. Search for the paper via web search. Confirm the *exact* title, *full* author list, year, venue, and identifier from the source. If you cannot find the paper, do not guess. Never write a citation from memory alone.

Failure mode: This commandment aims to address a well-known limitation of these systems: they hallucinate plausible but incorrect bibliographic entries.

Autonomy and Efficiency

A major problem we encountered was that, despite having a long todo-list of potential tasks and experiments, the agent consistently stopped to ask whether it should continue. The following two commandments aim at maximizing productive work within each session.

IV. Complete All Autonomous Work Before Reporting

Finish every task that does not need user input. Report once with all results. Never skip work because you estimate it "takes too long to implement."

Failure mode: The agent frequently stops to ask whether it should continue, even when the research plan specifies many more experiments that could be executed without additional input from the researcher. A related failure mode is that the agent often discards approaches because they "would take too long to implement" and potentially "only have modest impact". Modest impact aside, agents drastically underestimate their own coding speed; in fact, the implementation typically takes less than a minute. The only valid time concern is actual compute runtime measured in days.

V. Make It Work Before Moving On

An experiment crash is a bug, not a bad idea. Do not discard methods because of implementation failures. Investigate, fix, and re-run.

Failure mode: When encountering an implementation failure, agents often claim that the approach "doesn't work" and move on to an alternative. In practice, however, most of these crashes are simple bugs that can be fixed easily. For instance, when hitting an out-of-memory error, the agent concluded that the method "doesn't scale". Upon further investigation, it found an unnecessary materialization of a memory-intensive matrix, replaced it, and the method ran successfully, yielding significant improvements over the baseline.

Scientific Rigor

The following three commandments ensure that the agent follows the norms of scientific practice.

VI. One Variable per Experiment

Change exactly one thing per experiment. If two things change and the metric improves, you cannot know which helped.

Failure mode: If one experiment is successful and the agent has an idea for further improvement, it is often tempted to combine both the successful change and the new idea simultaneously in the next experiment. This makes it impossible to determine which change caused the improvement.

VII. Evaluate in Tiers

Tier 1 (seconds): does it run without crashing?
Tier 2 (minutes): any signal on a small subset?
Tier 3: full evaluation, i.e., the real metric that goes into the report. Use small-scale runs to catch bugs only. Never draw conclusions from small-scale results.

Failure mode: We want the agent to iterate quickly and distinguish between trivial and meaningful improvements. Consequently, we enforce that the agent (a) does not run a full, potentially expensive evaluation after every minor code change, and (b) does not discard ideas based on unsuccessful small-scale runs on toy problem instances.

VIII. Bound Your Expectations

Before implementing a heuristic, identify the theoretical best case, even if it is not realizable in practice. If you are "correcting" something, measure how much correction is theoretically possible.

Failure mode: To decide whether a method is successful, it is crucial to understand a theoretical upper bound on the possible improvement. The agent often observes a small improvement and reports it as a success, without assessing proximity to the theoretical maximum.

Documentation and Reproducibility

The following two commandments ensure that the agent documents its work reproducibly. This is one of the most important categories, as it enables restarting the research process from any given point.

IX. Record Everything

Every experiment gets a subsection in the report: goal, hypothesis, method, results table, analysis, next steps. Include failures. If it is not in the report, it did not happen. Visualize, don't just describe: create plots for distributions, comparisons, and scaling. Maintain `TODO.md` as a living checklist for open questions, unverified claims, and deferred work.

Failure mode: Without the rule, the agent runs experiments, observes results, and keeps them in its context window. As soon as this context window is compacted or cleared, the information is lost. At the same time, the strict rule "if it is not in the report, it did not happen" ensures that the agent does not mistakenly believe it has already obtained a result that was never recorded. Apart from the report, which we save as a LaTeX document, we also maintain a `TODO.md` file, which is equally critical, as it prevents the agent from forgetting about open questions, unverified claims, and deferred work.

X. Verify Before Claiming

Assume you are wrong until verified. Write verification scripts, not just explanations. Actively try to falsify your own claims, test edge cases, randomize inputs, search for counterexamples. Grade claims: *verified*, *partially verified*, or *unverified*.

Failure mode: Mathematical verification remains a major challenge for LLMs. We observed significant improvements when enforcing at least numerical verification of claims. For instance, the agent derives a formula whose derivation contains an error (e.g., a missing factor of two), but the results look plausible. A verification script that checks the formula against a brute-force computation on small instances catches this immediately and prevents the agent from continuing its argument

on a false premise. This active falsification, i.e., the process of deliberately trying to break your own hypothesis before confirming it, often reveals the key structural insight that makes the proof work.

Domain-Specific Commandments

The ten commandments above are intended to be universal. In addition, we found it beneficial to provide domain-specific commandments tailored to the research style of the domain, whether primarily theoretical or empirical. Beyond these broad categories, further specialization is useful: for instance, research in a specific subfield of mathematics benefits from commandments tailored to its particular challenges.

Domain: Compute-Intensive Research. For empirical projects involving GPU experiments, deep learning, or large-scale numerical simulations, we apply the following additional commandments:

- **One experiment per GPU; use them all (C1).** Check `nvidia-smi` before every batch of work. Assign each independent experiment to its own GPU. Never leave GPUs idle when independent tasks remain.
- **Context window hygiene (C2).** Prefer redirecting long-running output to log files and monitoring with `tail`. Only investigate logs in detail if something looks wrong.
- **Memory management (C3).** When observing out-of-memory (OOM) errors, do not conclude that the method "does not scale". Instead, systematically reduce memory: clear the GPU cache between experiments (`torch.cuda.empty_cache()`), enable gradient checkpointing, or process layers sequentially instead of in parallel. Print `torch.cuda.memory_summary()` to identify the allocation that causes the spike. Only after these mitigations fail is it valid to report a genuine scaling limitation.
- **Discover nodes first; dispatch independent experiments (C4).** When a multi-node Slurm allocation is active, discover available nodes at session startup and dispatch independent experiments to remote nodes via `remote-run`. Each dispatched job runs in its own container on the target node with full GPU access. Never dispatch dependent work: only experiments that are fully independent may run on remote nodes.

Domain: Mathematical Research. For theory-heavy projects involving proofs and derivations, we apply the following additional commandments:

- **Derivations before code (M1).** Write derivations step-by-step before implementing. Cross-reference paper equations. Before implementing a new method, search for prior work to flag potential rediscovery.
- **Precise notation (M2).** Use precise index notation (G_{jj}, not G_j, for diagonal elements of a matrix). Define all notation before first use; dimensions, ranges, scalar vs. vector vs. matrix. Apply the same rigor to negative results as to positive ones.
- **Counterexample-first reasoning (M3).** Before attempting a proof, actively search for counterexamples: randomize inputs, test boundary cases, enumerate small instances exhaustively. If a counterexample exists, the search finds it faster than a failed proof attempt reveals the obstruction. If no counterexample survives, the search often exposes the structural property that makes the proof work.

D. An Autonomous Session in Practice

E. Case Studies: Long-Form Versions

We present case studies demonstrating the framework across different research domains and integration levels. The first three (A–C) deal with LLM-related research questions: pretraining, pruning, and quantization. The remaining three (D–F) concern mathematical research: convex optimization, combinatorial optimization, and algebraic geometry. Each case study follows a consistent structure: domain, problem, what the agent did, results, and lessons learned. Throughout, we include figures, screenshots, and excerpts from the agent's reports as they were produced (indicated by a thin border); minor errors or rendering artifacts are preserved and marked with [*sic*] where appropriate.

```
  Let me now wait for the final check:

● Task Output bwzs8ty88
  1-hour mid-training status check
      Waiting for task (esc to give additional instructions)

✳ Gitifying… (8h 39m 19s · ↓ 39.3k tokens · thought for 20s)
└  Tip: Run /install-github-app to tag @claude right from your Github issues and PRs
─────────────────────────────────────────────────────────────────────────────────────
Background tasks
10 active shells

❯ sleep 2700 &&echo "≡ Final completion check ≡" &&python3 << 'PYEOF'…   (running)
  sleep 3600 &&echo "≡ 1h mid-training ≡" &&python3 -c "…   (running)
  sleep 5400 &&echo "≡ Final completion check ≡" &&for exp in E023_no_nor…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=6 uv run torchrun --standalone --npr…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=5 uv run torchrun --standalone --npr…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=4 uv run torchrun --standalone --npr…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=3 uv run torchrun --standalone --npr…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=2 uv run torchrun --standalone --npr…   (running)
  source ~/.bashrc &&CUDA_VISIBLE_DEVICES=1 uv run torchrun --standalone --npr…   (running)
```

Figure 7. A CLI agent during an autonomous research session: over 8 hours in, managing six parallel GPU training runs and three scheduled monitoring tasks. The same framework supports mathematical derivations, proofs, and verification alongside computational experiments. The agent is idle, consuming no tokens while waiting for a status check to complete.

E.1. Systematic Optimizer Exploration for LLM Pretraining

This case study demonstrates the framework's core experimental loop on a computationally intensive deep learning task: systematic, single-variable experimentation across a non-trivial optimizer design space, with multiple GPUs running independent experiments in parallel.

Domain and problem. AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017) has long been the dominant optimizer for language model pretraining. It maintains two buffers per parameter (first and second moments), requiring additional memory $2N$ compared to vanilla Stochastic Gradient Descent (SGD), where N is the number of parameters. The Muon optimizer (Jordan et al., 2024) takes a fundamentally different approach: instead of adaptive step sizes, it computes a momentum vector $M_t = \mu M_{t-1} + G_t$ and then applies Newton-Schulz (NS) orthogonalization to approximate $UV^\top$ from the Singular Value Decomposition (SVD) of the momentum buffer $M_t = U\Sigma V^\top$, so that $W_{t+1} = W_t - \eta \cdot \mathrm{NS}(M_t)$. This operation equalizes all singular values of the update and achieves strong results on LLM pretraining while using only N additional memory units (one momentum buffer) compared to SGD, half of AdamW's $2N$. A natural question arises: *can the spare N memory budget be exploited to make Muon better?* The agent was given this open-ended research question, the codebase of Semenov et al. (2025) as a standardized LLM pretraining benchmark (124M-parameter Llama on FineWeb, 10,000 iterations), and a multi-GPU compute allocation.

What the agent did. After establishing baselines (Muon, AdamW), the agent explored modifications to the Muon update rule, changing exactly one variable per experiment (Commandment VI). The central insight was that Muon converges faster when the vector it orthogonalizes is well-conditioned: normalizing the momentum buffer before orthogonalization means the same number of iterations yields a better update. The agent tested multiple normalization strategies, swept hyperparameters one at a time, and discovered two independent improvements: (1) a normalization technique applied before orthogonalization, and (2) the addition of weight decay to Muon's matrix parameters. Weight decay is a standard regularization technique and its benefit is not surprising in itself; however, the reference codebase implemented Muon without it, and because the agent tested each modification in isolation (Commandment VI), it was able to quantify this contribution separately and still identify the normalization improvement on top of it. A zero-overhead variant requiring no extra buffer was found to achieve nearly identical results. Following Commandment IX, each of the more than 40

experiments was documented in the agent's `report.tex` with goal, hypothesis, method, results table, and analysis.

The agent also identified several independent papers exploring normalization in the context of Muon: NorMuon (Li et al., 2025), AdaMuon (Si et al., 2025), and Muon+ (Zhang et al., 2026), each proposing a different normalization strategy. It implemented two of these methods in its codebase and ran a detailed comparison, analyzing the theoretical and empirical differences between the approaches (Commandment V). The existence of multiple concurrent works exploring the same design space underscores the need to carefully characterize how the agent's approach relates to and differs from each of them. While the agent conducted thorough literature searches, we cannot guarantee that its specific combination of modifications is truly novel. Accordingly, we keep the presentation at a high level and view these results primarily as initial directions to build on: the experiments are limited to a single architecture and dataset, and a full comparison across model scales, training setups, and concurrent methods would be necessary to draw any definitive conclusions. A standalone publication would further require a more in-depth prior-art investigation to establish precisely which aspects, if any, are new.

Results. Across more than 40 experiments documented in the agent's `report.tex`, the best configuration achieved a $\sim$5% improvement in validation perplexity over Muon (and $\sim$8% over AdamW) at the same $2N$ memory budget as AdamW. The two improvements are nearly additive: normalization alone provides $\sim$3%, weight decay alone $\sim$2%, and the combination $\sim$5% (Figure 8). The zero-overhead variant achieves $\sim$4.8% improvement at the *same* N memory footprint as baseline Muon, within a fraction of a perplexity point of the full method. Results were replicated across random seeds and a broader hyperparameter sweep.

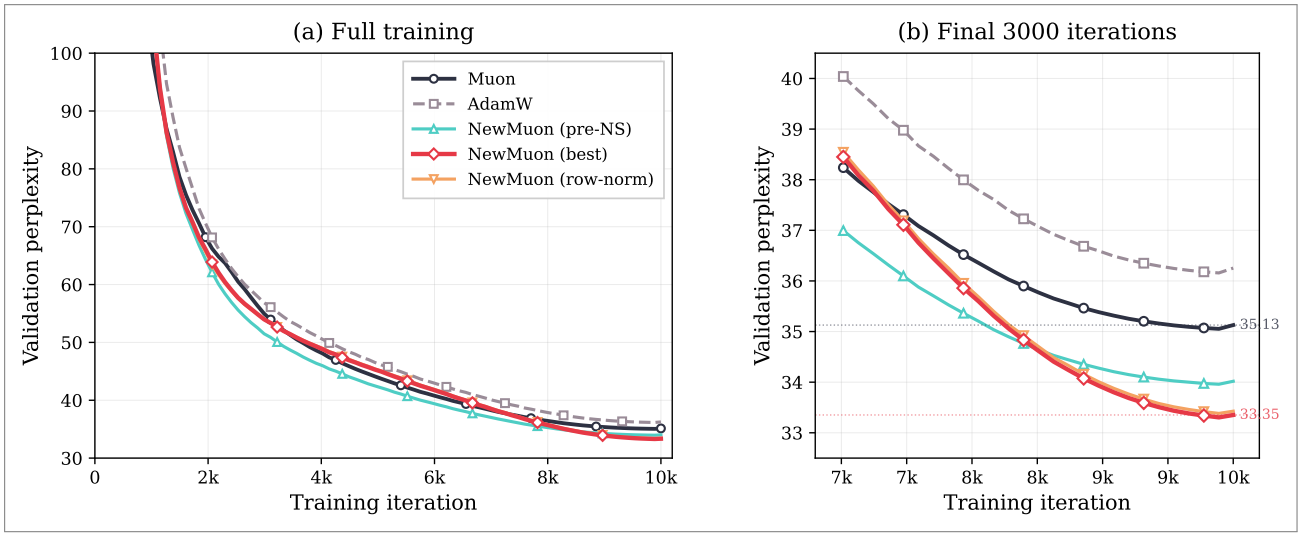

Figure 8. Training curves [*sic*] from the agent's report in Appendix E.1. Left: full training run. Right: final 3,000 iterations (zoomed). The agent's optimizer modifications consistently outperform both Muon and AdamW baselines throughout training, not only in the final iterations.

Lessons learned. The one-variable-at-a-time commandment (Commandment VI) was critical in this design space: the agent discovered that normalization and weight decay provide independent, nearly additive improvements only because it tested each in isolation before combining them. A 2×2 factorial ablation (normalization $\times$ weight decay) confirmed the near-additivity, which would have been obscured by testing them jointly from the start. An interesting aspect of the agent's research behavior is that, while the task explicitly granted an extra N memory budget, the agent proactively explored whether the same gains could be achieved without it, and found a zero-overhead variant that nearly matched the full method at the same N memory footprint as baseline Muon. The entire session ran for over twenty hours without human intervention. With multiple GPUs available, the agent ran independent experiments in parallel (one per GPU, Commandment C1); the framework's multi-node dispatch capability (Appendix C) enables large-scale concurrent experiments across compute nodes. Despite the long wall-clock time, actual token consumption remained modest: most time was spent waiting for training runs to finish while the agent redirected output to log files and monitored progress with lightweight commands (Figure 7), as encouraged by Commandment C2. The framework's emphasis on literature verification (Commandment III) prompted the agent to proactively search for related work, identify concurrent papers, and implement their methods for comparison.

While this is a useful first step, the limitations noted above show that such automated searches are not a substitute for the thorough prior-art investigation a human researcher would conduct before claiming novelty or asserting that the resulting method truly outperforms concurrent approaches.

E.2. Weight Reconstruction in Large Language Model Pruning

This case study illustrates a characteristic side effect of the agentic framework we propose: the agent was tasked with one research objective and *discovered* a different, more effective technique along the way (i.e., we observed *serendipity*).

Domain and problem. Pruning large language models (LLMs) reduces memory and compute costs by zeroing out weights, i.e., selecting a binary sparsity mask $M \in \{0,1\}^{d_{\text{out}} \times d_{\text{in}}}$ per weight matrix (cf., e.g., Zimmer et al., 2023a; Frantar & Alistarh, 2023; Sun et al., 2024). The constraints on M determine the sparsity pattern and, with it, the potential for hardware acceleration: unstructured sparsity removes arbitrary individual weights (Han et al., 2015; Zimmer et al., 2023b; 2024; 2025), while semi-structured patterns such as $N{:}M$ (Mishra et al., 2021; Zhang et al., 2023; Lasby et al., 2025) impose structure that is more amenable to hardware acceleration. The core challenge across all settings is *mask selection*: choosing which weights to zero out so that the pruned network's output remains close to the original (Roux et al., 2025; Zimmer et al., 2026). Once a mask is fixed, the pruned model's performance degrades compared to the dense original; one way to counteract this is *weight reconstruction*, i.e., adjusting the surviving weights to compensate for the removed connections (Frantar & Alistarh, 2023). Calibration data is drawn from C4 (Raffel et al., 2020); quality is measured by perplexity on the WikiText (Merity et al., 2016) test set (lower is better).

The project started with a concrete task: we had developed a pruning approach that aimed to find better masks, but it produced inconsistent results, sometimes failing catastrophically. The agent was provided with an existing codebase containing implementations of several pruning methods and the LaTeX derivation of our approach, and instructed to analyze why it failed, fix or replace the method, and empirically beat a set of baselines (Sun et al., 2024; Zhang et al., 2023) at 60% sparsity.

What the agent did. The agent first established that the existing approach was mathematically flawed and could not be repaired. While analyzing *why* it failed, the agent studied how pruning distorts the post-layer activations of each weight matrix and observed a severe imbalance: some rows lose over 50% of their activation-weighted output magnitude while others lose less than 10%. This byproduct of debugging led the agent to propose a simple post-pruning weight correction that restores the activation balance across rows and columns. Following Commandment VIII, the agent first computed an oracle bound via least-squares reconstruction to determine the theoretical limit, then validated the new method through the tiered evaluation protocol (Commandment VII) across five model scales.

Results. The method consistently reduces perplexity by 18–50% across five model scales (125M to 9B parameters), three architectures (OPT, Qwen, Gemma), and two pruning methods (RIA, Wanda). It requires only 10 lines of code, adds less than 1% computational overhead, and needs no hyperparameter tuning. The oracle comparison shows that this simple heuristic captures 92% of the improvement achievable by full least-squares reconstruction, leaving little room for more sophisticated approaches. Across 27 experiments documented in the agent's report, the improvements are robust and transfer to every model and pruning method tested. Figure 9 shows the scaling behavior across model sizes, reproduced [*sic*] from the agent's report; note, for instance, that the 50% sparsity line in the left panel ends at 1.5B because the agent found the 60% setting more promising and did not complete the remaining experiments.

Lessons learned. The original task was to fix a broken pruning mask; the actual outcome was a novel weight reconstruction method. The commandments forced the agent to analyze *why* the approach failed rather than simply trying the next idea, and this systematic analysis led to the discovery. Computing the oracle baseline (Commandment VIII) early on established that 92% of the theoretical optimum was already achieved, preventing wasted effort on a nearly closed gap. Finally, several extensions showed no benefit on small models but 7–11% improvement at 1.5–7B scale; the tiered evaluation protocol (Commandment VII) caught this systematically.

E.3. Column Ordering in LLM Quantization

This case study shows the framework operating as a systematic empirical researcher: given a well-defined design space, the agent mapped it comprehensively and discovered that the most important finding was not which method wins, but *when and*

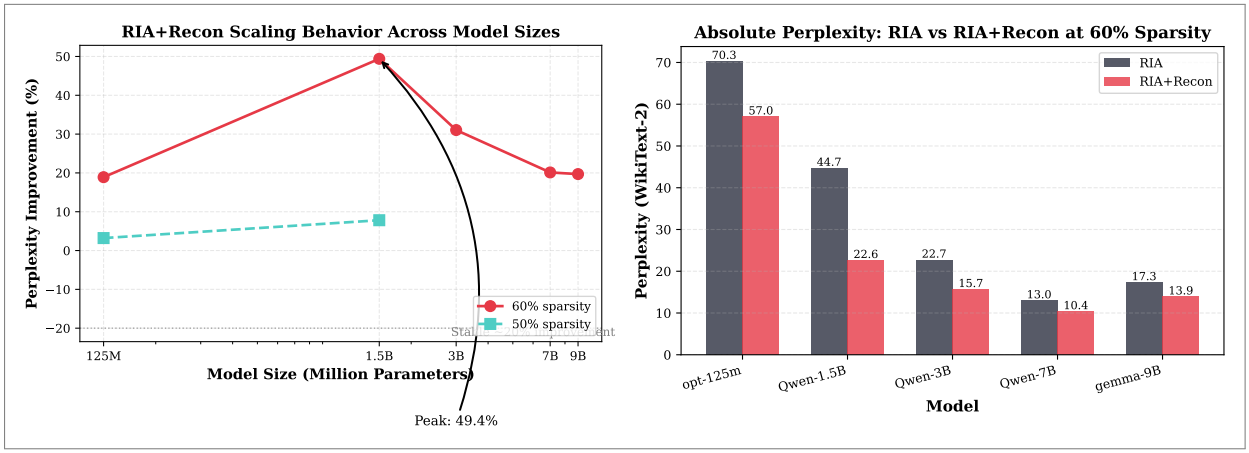

Figure 9. Plots [*sic*] from the agent's report for Appendix E.2, produced by the agent. Left: relative perplexity improvement vs. model size. Right: absolute perplexity comparison showing that the weight reconstruction method consistently outperforms the baseline across all tested model sizes.

why it matters.

Domain and problem. Post-training quantization compresses a pretrained language model by representing its weights in lower precision, substantially reducing the memory footprint and enabling deployment on consumer-grade hardware. GPTQ (Frantar et al., 2023), a widely used method, processes each weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ column by column to minimize the layer-wise reconstruction error $\|(W - \hat{W})X\|_F^2$, where $\hat{W}$ denotes the quantized matrix and $X \in \mathbb{R}^{d_{\text{in}} \times n}$ are calibration activations. Each column's rounding error is propagated to subsequent columns via the inverse of the Hessian $H = 2XX^\top \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$. The order in which columns are processed affects the final quality. A post-publication variant known as "act-order"[2] sorts columns by descending Hessian diagonal, with the intuition that high-sensitivity columns benefit from having more subsequent columns available for error compensation. The agent was tasked with investigating whether better orderings exist, how the effect depends on model architecture, and validating findings across model families. Calibration data is drawn from C4 (Raffel et al., 2020); quality is measured by perplexity on the WikiText (Merity et al., 2016) test set (lower is better).

What the agent did. The agent began with a mathematical analysis of why column ordering matters, then implemented and compared seven ordering strategies, first on single weight matrices, then at full model scale. Following Commandment X, it created verification scripts for all error propagation and refinement formulas before running any benchmarks (Figure 10). Cross-architecture validation (Commandment VII) across five model families (Qwen, Llama, Gemma, Mistral, Yi) revealed the central finding: the ordering effect varies by more than two orders of magnitude across architectures.

Results. Column ordering is the single most impactful improvement to GPTQ, but its magnitude is entirely architecture-dependent: it reduces perplexity by 74% on Llama-3.1-8B but only 0.1% on Gemma-2-9B at 4-bit. This finding would have been missed without systematic multi-architecture validation: on Qwen-1.5B alone, the effect is 20%, giving no indication that it ranges from 0.1% to 74% across architectures. Among the seven ordering strategies tested, alternatives that incorporate the quantization error magnitude alongside column sensitivity occasionally outperformed act-order (e.g., at 3-bit on certain architectures), but no single strategy dominated consistently across all architectures and bit widths. Nine of the 24 experiments produced negative results, each documented with the same rigor as positive ones (Commandment IX): many approaches failed because GPTQ's error propagation via Ordinary Least Squares (OLS) already minimizes the correlations these methods would exploit. A critical implementation bug in group quantization was caught because the agent investigated a failure rather than abandoning the method (Commandment V): pre-computing scale parameters from initial instead of error-propagated weights produced catastrophic results (perplexity 437 vs. 9.22 after the fix). The agent's report documents all 24 experiments and 11 key findings.

[2]Commit `a4c3c89`, March 2023, in https://github.com/IST-DASLab/gptq.

Verification

What: $GPTQ_{new}$ error propagation and refinement formulas
Method: Numeric tests on small matrices (32×64, 32×128)
Script: `scripts/verify_gptq_new.py`
Outcome: All 5 tests pass. No bugs found. One known approximation documented (within-chunk propagation in $GPTQ_{new}$).

Status: Complete.

Figure 10. A screenshot [*sic*] from the agent's report in Appendix E.3. Before running any benchmarks, the agent audited all error propagation and refinement formulas through numeric tests on small matrices (Commandment X).

Lessons learned. The negative results (9 of 24 experiments) were more informative than the positive ones: each failure clarified *why* simpler methods work, revealing that GPTQ's OLS-based error propagation already handles what sophisticated alternatives attempt. With four GPUs, the agent ran independent model evaluations in parallel (one per GPU, Commandment C1), efficiently covering five model families with multiple configurations each. The "Make It Work" commandment (Commandment V) prevented a false negative: group quantization initially appeared broken on Llama, but investigation revealed a subtle implementation bug whose fix turned a catastrophic failure into the best result.

E.4. Tight Lower Bounds for Frank–Wolfe on Uniformly Convex Sets (Full Version of Section 4.1)

This case study demonstrates the framework on a problem in convex optimization, where the agent's primary output is the proof of a new theorem. Unlike the computational and empirical case studies, the research here required sustained interaction between numerical exploration and theoretical development: the agent discovered the correct proof strategy through systematic experimentation before formalizing it.

Domain and problem. The Frank-Wolfe (FW) algorithm minimizes a smooth convex function over a convex constraint set using only a linear minimization oracle (LMO). On strongly convex sets, the known $\mathcal{O}(1/T^2)$ upper bound was recently shown to be tight: Halbey et al. (2026) gave a lower bound for vanilla FW in dimension 2 by analyzing the dynamics of the iterates on a worst-case instance. Shortly after, Grimmer & Liu (2026) proved an information-theoretic lower bound in the high-dimensional setting for a broad class of LMO-based algorithms. For uniformly convex sets of order $p > 2$ (e.g., ℓ_p-balls), Kerdreux et al. (2021) established an upper bound of $\mathcal{O}(1/T^{p/(p-1)})$, but no matching lower bound was known. The goal was to prove lower bounds for the uniformly convex setting based on the techniques used by Halbey et al. (2026) or Grimmer & Liu (2026).

What the agent did. The agent began by studying both existing lower-bound techniques and attempting to generalize the high-dimensional construction by Grimmer & Liu (2026) to ℓ_p-balls. This did not succeed: the construction relies on decomposing strongly convex sets as intersections of shifted Euclidean balls, and the agent did not find a direct analogue for uniformly convex sets of order $p > 2$. Following Commandment IX, the agent documented this negative result and pivoted to the alternative approach of Halbey et al. (2026), which analyzes the FW iterates directly on a worst-case instance. The agent derived the FW dynamics on ℓ_p-balls in closed form and verified each component numerically (Commandment X). Experiments across multiple values of p revealed that the iterates alternate in sign and settle onto a low-dimensional curve whose shape can be characterized analytically, which suggested the right proof strategy. The agent first estimated the key constants numerically, then derived them in closed form, and finally assembled a rigorous proof for $p \geq 3$ with explicit convergence rates. Each proof step was accompanied by Julia verification scripts using `BigFloat` arithmetic, totaling over 30 individual checks. The case $p \in (2, 3)$ was identified as qualitatively different: sign alternation breaks down intermittently, and the proof technique does not apply.

Results. The main result establishes a lower bound of $\Omega(1/T^{p/(p-1)})$ for vanilla FW on p-uniformly convex sets for any $p \geq 3$, matching the upper bound of Kerdreux et al. (2021) and resolving the open question for this regime. The proof

provides explicit convergence constants, all verified numerically to $< 0.2\%$ relative error. The case $p \in (2, 3)$ remains open: numerical evidence supports the same rate, but the proof technique does not extend.

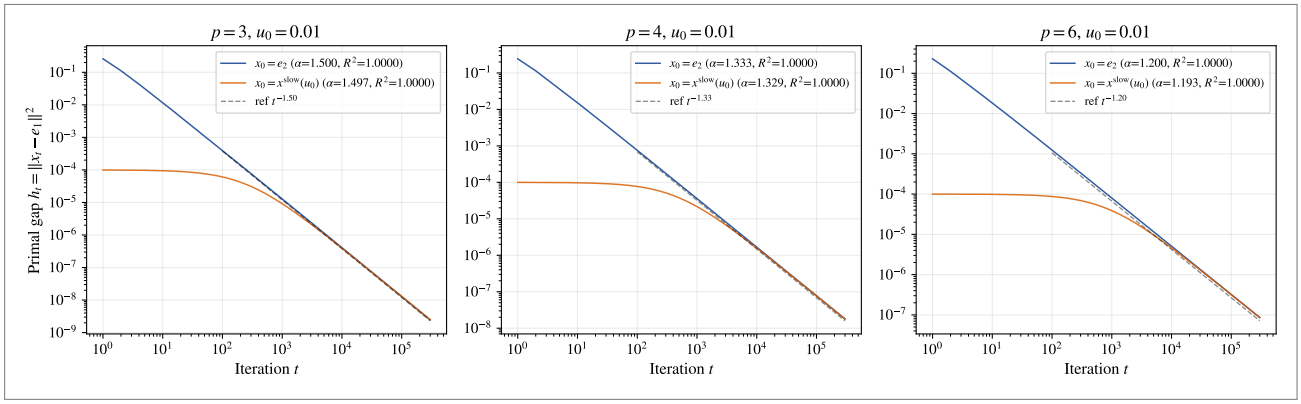

Figure 11. A plot [*sic*] from the agent's report: log-log convergence of $\|x_t - e_1\|^2$ for $p \in \{3, 4, 6\}$ starting from $x_0 = e_2$ (blue) and from $x_0 = x_0^{\text{slow}}(10^{-2})$ (orange), where x_0^{slow} is the worst-case initialization from the proof and α is the fitted coefficient of $t^{-\alpha}$.

Lessons learned. The correct proof strategy emerged from the agent's numerical exploration: patterns observed in the iterates suggested the right analytical approach, and the key constants were first estimated computationally before being derived in closed form. This "conjecture from computation, then prove" loop, enabled by the framework's emphasis on creating verification scripts alongside every mathematical claim (Commandment X), is a natural workflow for this type of problem. The failed generalization of Grimmer & Liu (2026) was equally informative: it helped us understand which parts of the proof are hard to extend to the uniformly convex setting, guiding the pivot to the successful approach. Following Commandment IX, this negative result was documented thoroughly.

E.5. Multi-Variable Dual Tightening for Mixed-Integer Optimization

This case study demonstrates the framework in combinatorial optimization. Its main contribution is a multi-variable generalization of dual tightening, together with a prototype implementation in the Boscia solver. The case study spans the full research cycle: deriving the result, proving it, implementing it, and evaluating it computationally.

Domain and problem. Boscia (Hendrych et al., 2025) is a Frank-Wolfe-based branch-and-bound solver for mixed-integer nonlinear optimization over polytopes ($\min_{x \in X \cap \mathbb{Z}^J} f(x)$ with f smooth convex), where $X \subseteq \mathbb{R}^n$. A key pruning mechanism is *dual tightening*. At a relaxed solution x^t with gradient $g = \nabla f(x^t)$ and Frank-Wolfe dual gap $\gamma(x^t) = \max_{v \in X} \langle g, x^t - v \rangle$, convexity implies that any feasible point $x \in X$ with objective value at most some upper bound UB (e.g., from an incumbent) satisfies $g_j(x_j - \ell_j) \leq \text{RHS}$ for each variable j, where $\text{RHS} := \text{UB} - f(x^t) + \gamma(x^t)$ and ℓ_j is the lower bound of x_j. This allows variables to be fixed one at a time. The project investigated whether this extends to *subsets*: for a set S of variables at their lower bounds, $\sum_{j \in S} g_j(x_j - \ell_j) \leq \text{RHS}$, so when the combined gradient contribution exceeds the budget, a *conflict constraint* prevents all variables from simultaneously deviating from their current bounds. For binary variables, a pairwise conflict $g_i + g_j > \text{RHS}$ implies $x_i + x_j \leq 1$ (a conflict graph edge); higher-order conflicts (triples, quadruples) capture interactions that pairwise constraints miss. The goal was to derive the mathematical result, implement it as a conflict graph with constraint propagation integrated into Boscia via callbacks, and benchmark on a diverse set of Mixed-Integer Nonlinear Programming (MINLP) instances.

What the agent did. The agent started from Boscia's existing single-variable dual tightening result (Theorem 3 of Hendrych et al. (2025)), identified the natural generalization via the convexity inequality, and formulated and proved a multi-variable dual tightening theorem with corollaries for pairwise and higher-order binary conflicts. Before implementation, the agent first tried to verify the proof both symbolically, using `Symbolics.jl` with 2,387 checks, and numerically, using a script that exhaustively enumerated all 2^n feasible points for small instances (487 checks). This verification caught an error in the initial derivation: the bound for the at-least set constraint had been inverted, which would have led to overly aggressive fixings for upper-bound variables. The agent then implemented a `ConflictGraph` data structure with constraint propagation and integrated it into Boscia via two *callbacks* (Figure 12), requiring no source modifications beyond fixing a pre-existing `Dict`

type bug. A key design decision made by the agent was to derive conflicts only at the root node. Because these conflicts use the global Frank-Wolfe gap, they remain valid throughout the search tree, but are more conservative than conflicts derived locally at child nodes. The agent also explored tighter child-node conflicts, but early tests suggested that the additional overhead and numerical instability were not worth the potential gain.

Results. Across 33 instances in six problem categories ($n = 12$ to $n = 300$, 10-minute time limit), partition-constrained instances show the strongest improvement (up to 18.9% node reduction, from 127 to 103 nodes on a 48-variable instance), where partition constraints create tight cross-block coupling that the conflict graph captures. The root-only design is deliberately conservative, and most instances show 0% node reduction because the root budget is loose. However, this guarantees correctness, which is critical for an exact mixed-integer convex optimization solver, and all 33 instances produce identical optimal objectives in both modes. As expected, separable quadratic instances show no benefit because diagonal objectives create no cross-variable coupling, confirming the theoretical prediction.

4.1 Callback architecture

The conflict graph is integrated via two standard Boscia callbacks. No Boscia source modifications are required beyond the existing Dict type fix for `settings.tightening` (commit `c8f86437b`).

`propagate_bounds(tree, node)`
: Called at each node *before* the Frank-Wolfe solve. Propagates conflict-implied fixings from the root-derived conflict graph into `node.local.bounds`, rebuilds the LMO, and cleans the active set (see Section 4.2).

`bnb_callback(tree, node)`
: Called *after* each node is processed. At the root (`node.std.id` $= 1$): derives conflicts into the global graph and stores a gradient/iterate snapshot for re-scanning. At non-root nodes: checks whether the incumbent improved and, if so, re-scans the root snapshot with the tighter RHS $= \mathrm{UB}_{\mathrm{new}} - f(x^t_{\mathrm{root}}) + \tau \cdot \gamma_{\mathrm{root}}$.

Figure 12. A screenshot [*sic*] from the agent's report: the callback architecture in Appendix E.5. The conflict graph is integrated into Boscia via two standard callbacks, `propagate_bounds` (before each Frank-Wolfe solve) and `bnb_callback` (after each node), without modifying Boscia's source code.

Lessons learned. This case study shows that the framework is effective for projects that combine theorem proving, verification, implementation, and experiments in a single workflow. The verification-first approach (Commandment X) was crucial for the overall correctness. It caught the inverted at-least bound bug before it entered the experiment phase. The negative results were useful as well. The lack of improvement on separable instances matched the theory, while the $26\times$ overhead on a sparse regression instance with 150 indicator variables exposed a concrete bottleneck and pointed to straightforward fixes, including better data structures and a cap on propagated conflicts. Following Commandment IX, these outcomes were all documented in the report, which made the evaluation more transparent and more useful for guiding future improvements.

E.6. Finding Maximal Real Solutions in K_7 Power Networks

This case study shows the framework operating as a computational scientist for discovery. Starting from a published method for characterizing typical behavior, the agent reconstructed the pipeline and repurposed it for directed extremal search, discovering an improved lower bound.

Domain and problem. Electrical power grids can be modeled as networks of buses connected by transmission lines, where the physics imposes a system of polynomial equations whose real solutions correspond to feasible operating states. Solutions to these power flow equations define the operating points of the network and underpin decisions ranging from long-term planning and capital investment to day-to-day resource scheduling, market operations, and real-time stability analysis. The equations depend on tunable parameters (susceptances), which appear as coefficients in the system. This motivates a natural structural question, raised explicitly by Lindberg et al. (2020): *for a fixed network topology, what is*

the maximum number of feasible operating states over all parameter choices? Lindberg et al. (2020) characterized the *distribution* of solution counts for several topologies, including K_7 (seven buses, every pair connected), using a continuation pipeline orders of magnitude faster than naive solving. However, they did not target extremal instances, i.e., those with a maximal number of real solutions, explicitly. Our goal is therefore to adapt the sampling technique from Lindberg et al. (2020) to identify parameter settings that yield extremal instances.

What the agent did. The agent first reconstructed the pipeline of Lindberg et al., which was a nontrivial task. Reproducing the published results required several rounds of refinement to align the implementation with the paper's symmetry conventions, parameterization choices, and solution-counting bookkeeping. Once this baseline was validated, the agent adapted the pipeline from sampling to extremal search. To explore the parameter space effectively, the agent combined several heuristic search strategies, including hill climbing, simulated annealing, and warm starts from the best susceptance vectors found so far. These methods were used iteratively to bias the search toward regions of parameter space with unusually large numbers of real solutions, with each successful run informing the next.

Results. Random sampling of 1.4 million parameter vectors, following the original paper's sampling protocol, found at most 120 (nontrivial) feasible states. Targeted search instead identified a parameter vector with 192 feasible states. The agent also perturbed this parameter vector to verify that the 192-solution count is not confined to an isolated parameter point, but persists in a neighborhood of parameter space. The maximum real solutions problem for K_7 remains open. However, adapting Lindberg et al.'s continuation pipeline for extremal search yields a substantially stronger computational lower bound.

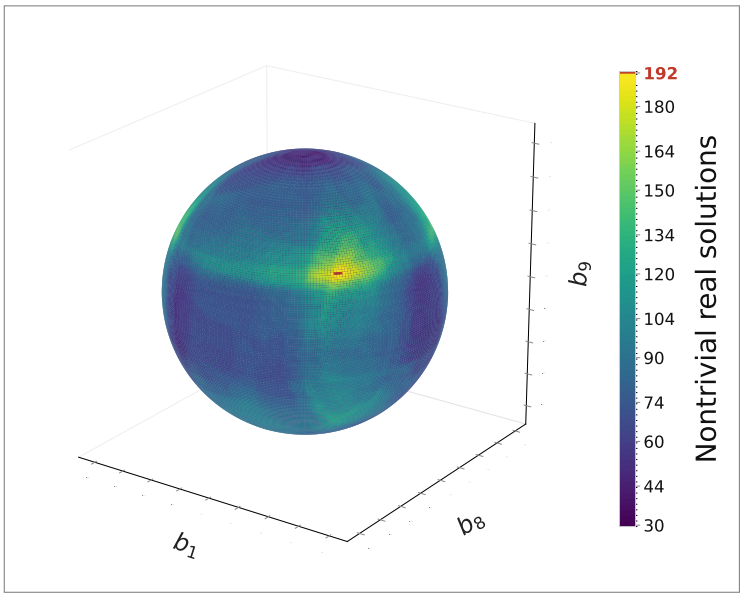

Figure 13. A plot [*sic*] from the agent's report: a three-parameter slice of the 21-dimensional K_7 susceptance space, obtained by varying b_1, b_8, and b_9 while fixing the remaining 18 parameters at the values of the best-found instance. Each point is colored by the number of nontrivial feasible operating states.

Lessons learned. This case study highlights the importance of verifiable intermediate artifacts: published tables and solution-count distributions were essential for checking that the reconstructed pipeline matched prior work before launching the extremal search (Commandment X). It also underscored the value of staged evaluation (Commandment VII): because individual searches can run for hours, the agent benefited from first validating correctness on cheaper checks and only then scaling up to long-running optimization runs. More broadly, the study shows that the agent need not rely on an existing codebase to begin exploration.

F. Discussion: Extended Limitations and Future Directions

The main paper's discussion is intentionally brief; we expand it here with the limitations and future directions that informed the design of our framework and that we believe matter most for the broader community.

Workflow design over model capability. A central claim of this paper is that effective agentic research does not require a specialized system built from scratch. Instead, it can be built around existing general-purpose agents, provided they are embedded in a disciplined and inspectable workflow. In our setup, the agent operates with persistent instructions, a sandboxed environment, written progress reports, `TODO.md` files, and a small set of methodological rules: change one variable at a time, evaluate in stages, and verify results before reporting them, among others. In practice, these additions were sufficient to extend the agent from a tool for isolated coding tasks into a useful research collaborator for exploratory and implementation-heavy work. Our experience suggests a simple conclusion: model capability matters, but workflow design matters just as much. These systems are only useful when their outputs can be checked and their intermediate steps revisited. This keeps the researcher responsible for direction, judgment, and verification, even when substantial exploratory or technical work is delegated.

Verification in the loop, not at the end. At the same time, this approach does not eliminate the need for expert oversight or final verification. In our framework, however, oversight is not reserved only for the end of the process; it is built into the workflow itself. A central requirement is that the agent must be able to test, challenge, and potentially refute its own claims through staged evaluation, intermediate checks, and explicit internal validation procedures. In our experience, these internal verification mechanisms are crucial. Without them, experiments can easily become structured to simply confirm an initial hypothesis. Final expert verification remains necessary, but it is far more reliable when supported by a workflow that already produces inspectable and continuously tested intermediate results. We emphasize that the case studies and reports do not constitute finished papers that are ready for publication, but rather records of meaningful research progress.

Limitation: verification. A fundamental limitation of our framework, shared with other agentic systems, is result verification. Natural-language proofs remain difficult to verify and require manual inspection. While code is usually easier to check, subtle implementation errors can still invalidate conclusions. Citations must also be verified carefully, since agents may hallucinate references or bibliographic details. This is not only a technical limitation but also a matter of responsible use: researchers must invest substantial effort in verifying agent outputs, especially because such outputs may be even harder for others to assess independently. As Su (2022) argue, researchers are often the best reviewers of their own papers; likewise, we argue that they are ultimately responsible for verifying the work produced by their agents.

Limitation: context. Long experimental sessions with many runs and large outputs can exceed a model's context window and trigger compaction. Because compaction is inherently lossy, the agent may forget details from earlier in the session, revisit failed approaches, or miss important observations. Practical mitigations include routing long outputs to log files and monitoring them with `tail`, manually invoking compaction commands such as `/compact`, and relying on persistent artifacts such as `report.tex` and `TODO.md` as re-entry points and external memory. We also tested autonomous compaction, but found it to have no positive impact. Robust context management remains an open challenge.

Limitation: cost. Long autonomous sessions with frontier models can incur nontrivial API costs. In practice, however, these costs are often relatively small since much of the wall-clock time in Level 4 sessions is spent waiting for experiments to finish rather than generating tokens. Still, cost remains a meaningful limitation, particularly for long-running studies and large-scale evaluations.

Future direction: extension to other domains. While our paper focuses on the application of our framework to machine learning and mathematical research, in principle it could be applied more broadly to other disciplines, such as physics, chemistry, economics, or the social sciences. Adapting the framework to these settings would require domain-specific tools, evaluation protocols, and safety checks, but the general paradigm of iterative experimentation, artifact management, and human verification may transfer well beyond our current case studies.

Future direction: more robust memory. Another important direction is improving how the system stores, retrieves, and updates information over long research sessions. Better memory mechanisms could help agents maintain continuity across

experiments, avoid revisiting failed approaches, and make more effective use of prior observations. This would be especially valuable for complex projects that unfold over many iterations and generate substantial intermediate state.

Future direction: multi-user collaboration. Our setup is currently designed for a single user interacting with a single main agent. An important future direction is extending this setting to support collaboration among multiple users, multiple agents, or both. Such a setting raises new challenges in coordination, communication, provenance tracking, and conflict resolution, but it could also make agentic research workflows more effective for team-based projects.

G. Related Work

We survey three bodies of work: AI systems that produce mathematical results autonomously (Appendix G.1), research on mathematicians actively using AI in their workflow (Appendix G.2), and agentic frameworks for scientific discovery (Appendix G.3). For broader surveys of AI for mathematics and scientific discovery, we refer to Ju & Dong (2026), Carbone (2025), and Zheng et al. (2025b).

G.1. AI Generating Mathematics

Competition-level mathematics. In recent years, progress in AI mathematical reasoning has been especially visible in competition-level mathematics, where performance is relatively easy to compare because problems typically have a single, closed-form final answer that can be scored automatically.[3] Early results came from specialized systems: AlphaProof (Hubert et al., 2025) combined reinforcement learning with the Lean proof assistant to reach silver-medal performance at the 2024 IMO, while AlphaGeometry (Trinh et al., 2024) and AlphaGeometry2 (Chervonyi et al., 2025) paired a neural model with a symbolic deduction engine to achieve gold-medalist performance on historical olympiad geometry. More recently, the emphasis has shifted toward off-the-shelf frontier models strengthened by verification and refinement: Huang & Yang (2025) report a model-agnostic pipeline that, with Gemini 2.5 Pro, Grok-4, or GPT-5, solves five out of six problems on the 2025 IMO under contamination-avoiding protocols. In parallel, proprietary systems such as Aristotle (Achim et al., 2025) combine informal reasoning with formal verification to achieve gold-medal-equivalent performance on the 2025 IMO. Finally, the same verification-first approach is now claimed at the undergraduate level: AxiomMath (2025) reports that AxiomProver produced Lean-checked solutions to all Putnam 2025 problems (a perfect $120/120$).[4] To move beyond competition-style evaluation, recent benchmarks increasingly probe research-level questions arising in active mathematical workflows, such as the encrypted, author-curated problem set in *First Proof* (Abouzaid et al., 2026).

Constructions and algorithms. Beyond proving theorems, AI has generated novel mathematical constructions and faster classical algorithms by *searching over programs*: an LLM proposes candidate code, an automated evaluator scores it, and an iterative loop improves the best candidates. FunSearch (Romera-Paredes et al., 2024) introduced this template, yielding new constructions for the cap set problem and improved online bin packing heuristics. AlphaEvolve (Novikov et al., 2025) scales the same evolutionary idea; in large-scale mathematical experiments it rediscovered best-known solutions across 67 problems and improved several, including autocorrelation inequalities (Georgiev et al., 2025). Recent open-source works have proposed methodological extensions, including OpenEvolve, ShinkaEvolve, ThetaEvolve, DeltaEvolve, and AdaEvolve (Sharma, 2025; Lange et al., 2025; Wang et al., 2025c; Jiang et al., 2026; Cemri et al., 2026). Most such systems are *closed-loop* and largely *non-interactive*: progress comes from automated propose–evaluate iterations rather than back-and-forth dialogue with a human. Related approaches have also produced faster algorithms: AlphaTensor (Fawzi et al., 2022) discovered efficient tensor decompositions for matrix multiplication, and AlphaDev (Mankowitz et al., 2023) found improved sorting routines now deployed in production software.

Data-driven and learning-augmented mathematics. A complementary line of work uses AI to generate candidate mathematical objects from data, whose correctness is then verified either automatically (via symbolic or optimization-based methods) or by human experts. Examples include data-driven conjecturing and candidate filtering (Davies et al., 2021; Mishra et al., 2023; Chuharski et al., 2024), learning-augmented Lyapunov, Sum-of-Squares, and Border basis pipelines (Alfarano et al., 2024; Zou et al., 2025; Pelleriti et al., 2025; Kera et al., 2025), neural-guided discovery of six-colorings for the Hadwiger–Nelson problem (Mundinger et al., 2024; 2025), and ML+high-precision optimization uncovering unstable self-similar solutions in fluid dynamics (Wang et al., 2025b). Symbolic regression further extracts interpretable laws from

[3] Correct final answers need not imply correct proofs (Dekoninck et al., 2026).

[4] cf. https://axiommath.ai/territory/from-seeing-why-to-checking-everything

data (Udrescu & Tegmark, 2020; Ruan et al., 2026).

Formal theorem proving and autoformalization. A rich ecosystem of LLM-based formal proving tools has emerged around *Lean 4* (de Moura & Ullrich, 2021). LeanDojo (Yang et al., 2023) provides an interface to Lean proof states and retrieval over mathlib (mathlib Community, 2020), while Lean Copilot (Song et al., 2025) integrates LLM assistance into the Lean workflow. Dedicated provers include DeepSeek-Prover (Xin et al., 2024), which leverages large-scale synthetic proof data, and DeepSeek-Prover-V2 (Ren et al., 2025), which adds reinforcement learning with explicit subgoal decomposition and introduces ProverBench for evaluation. Goedel-Prover-V2 (Lin et al., 2025) scales expert iteration with scaffolded data synthesis and verifier-guided self-correction. Complementary directions focus on knowledge reuse and structured reasoning: LEGO-Prover (Wang et al., 2023) builds and reuses a growing library of verified lemmas, while Hilbert (Varambally et al., 2025) connects informal reasoning with formal verification through recursive decomposition. TheoremLlama (Wang et al., 2024) and Mathesis (Xuejun et al., 2025) explore adapting general-purpose models and end-to-end pipelines from natural language to Lean proofs. Recent *agentic* frameworks emphasize tool use and iterative compiler-feedback loops rather than one-shot generation: APOLLO (Ospanov et al., 2025) performs modular proof repair and sub-lemma isolation, Ax-Prover (Breen et al., 2025) uses multi-agent tool-based proving across scientific domains, and LeanAgent (Kumarappan et al., 2025) studies continual adaptation across evolving repositories. In a different direction, LeanProgress (George et al., 2026) guides search by predicting proof progress to improve performance on long proofs. On the data side, MUSTARD (Huang et al., 2024) generates uniform theorem-and-proof training data with formal verification. For evaluation, miniF2F (Zheng et al., 2022) and PutnamBench (Tsoukalas et al., 2024) provide competition-style benchmarks, while SorryDB introduces a dynamically updating stream of open `sorry` tasks mined from real-world Lean projects, mitigating contamination. Autoformalization, i.e., translating informal mathematics into machine-checkable form, was shown to be feasible with LLMs by Wu et al. (2022). Recent work addresses this through dependency-graph decomposition (Wang et al., 2025a), chain-of-states proof translation (Wang et al., 2025d), and evaluation on real-world mathematical definitions (Zhang et al., 2025b). Agentic end-to-end pipelines such as MerLean (Ren et al., 2026) extend this to scientific domains. We refer to Weng et al. (2025) for a comprehensive survey.

Frontier systems and research-level evaluation suites. Beyond competition benchmarks, several recent efforts target *research-level* mathematics. *First Proof* (Abouzaid et al., 2026) introduces an author-curated set of ten questions arising naturally in the authors' research, with answers not publicly released. Other benchmarks include continuously refreshed collections drawn from arXiv papers (RealMath (Zhang et al., 2025a)) and curated sets of exceptionally challenging, unpublished problems reviewed by domain experts (FrontierMath (Glazer et al., 2025)). Aletheia was evaluated directly on *First Proof*: roughly three weeks after the challenge was introduced, Feng et al. (2026a) report that Aletheia autonomously solved six out of ten problems. Notably, some of these results are now accompanied by machine-checked proofs: for example, Sothanaphan (2026) provide a Lean formalization of a resolution of an Erdős problem attributed to Achim et al. (2025).

G.2. Mathematicians Using AI

Frameworks and perspectives. The literature on AI and mathematical practice is broad, so we highlight only those lines of work most directly relevant to our framework. Haase & Pokutta (2026) propose four levels of human-AI co-creativity: Digital Pen, AI Task Specialist, AI Assistant, and AI Co-Creator. These categories provide a conceptual vocabulary that we build on in Appendix A. Their treatment is intentionally broad and domain-agnostic, serving primarily as a conceptual template to which domain-specific details can be added. Henkel (2025) offer a complementary perspective from mathematics, arguing that AI should augment rather than replace mathematical reasoning and proposing five guiding principles for its responsible use. Noorani et al. (2025) formalize the complementary strengths of humans and AI in uncertainty quantification, providing theoretical guarantees for collaborative prediction. Most recently, Avigad (2026) consider recent developments in AI-driven mathematics and argue that mathematicians should remain actively involved in the use of these systems. Our work shares these perspectives but addresses a different question: given these emerging capabilities, how should a working researcher use them in practice?

Documented case studies. Over the past several months, a growing number of papers have documented how mathematicians interact with chat-based AI systems to obtain new research results (Bubeck et al., 2025; Diez et al., 2025; Alexeev & Mixon, 2026; Ivanisvili & Xie, 2025; Feldman & Karbasi, 2025; Salim, 2025; Dobriban, 2025; Schmitt, 2025). More specialized agentic systems with varying degrees of autonomy are also being developed (Liu et al., 2025; Feng et al., 2026b)

and have already produced new mathematical results (Lee & Seo, 2026; Feng, 2026). AI coding agents provide yet another pathway by enabling large computational searches: Knuth (2026) report that Claude solved an open Hamiltonian cycle decomposition problem through iterative exploration. These examples likely represent only a small fraction of emerging workflows.

G.3. Agentic Research Frameworks

Automated scientific discovery. Lu et al. (2024) introduced *The AI Scientist*, an end-to-end system that generates hypotheses, runs experiments, and writes papers; its successor (Yamada et al., 2025) reported an AI-generated paper accepted through peer review. Subsequent systems explore adjacent design points, from semi-automated, code-centric experimentation (CodeScientist (Jansen et al., 2025)) to end-to-end agent pipelines that incorporate explicit mechanisms for human feedback and cumulative reporting (Schmidgall et al., 2025; Schmidgall & Moor, 2025). AlphaApollo (Zhou et al., 2025) combines multi-turn tool use, reinforcement learning, and iterative evolution with tool-assisted verification, showing improved performance on several mathematical reasoning benchmarks. As these pipelines grow more complex, rigorous *benchmarking* has emerged as a central challenge, with recent work proposing evaluations that target both full workflows and their individual steps (Chen et al., 2025; Bragg et al., 2025). Taken together, these works highlight a common requirement: agent outputs must be checkable (e.g., as code, logs, or derived claims) and include explicit points for verification and human steering, rather than being treated as opaque end-to-end generations. Karpathy's *autoresearch* exemplifies a minimalist variant: an agent iteratively modifies a single file, runs fixed-budget training, and keeps or discards based on validation performance (Karpathy, 2026). Our framework targets the complementary regime of multi-file, multi-objective research with structured reporting and verification. For broader context, we refer to recent surveys (Ferrag et al., 2025; Zheng et al., 2025a).

Agentic coding tools. Terminal-based coding agents such as Claude Code, OpenCode, Codex CLI, and Gemini CLI (Anthropic; Anomaly; OpenAI; Google) extend AI assistance beyond chat by enabling users (Handa et al., 2025-12-04, 2025) (software engineers, analysts, and researchers alike) to delegate work within a persistent local project. These agents can read and edit files and invoke development tools (e.g., shells, test runners, linters, and formatters) from within a CLI interface, producing inspectable artifacts such as patches, diffs, and test outputs. This inspectable, file-based workflow is central to our setting: it enables reproducible iteration and makes it possible to attach verification hooks (tests, proofs, consistency checks) directly to the agent's actions. A key recent development is the growth of long-running autonomy: in Claude Code, the 99.9th-percentile turn duration nearly doubled from under 25 to over 45 minutes between October 2025 and January 2026 (McCain et al., 2026), reducing the need for constant supervision while increasing the importance of robust guardrails. Finally, these tools separate the underlying model from a repository-scoped instruction file, allowing us to express our framework as a portable, model- and harness-agnostic procedure that applies across Claude Code, OpenCode, Codex CLI, and related CLI agents.

H. Reproducibility

The entire workflow is distributed at github.com/ZIB-IOL/The-Agentic-Researcher. The repository contains: (i) the `INSTRUCTIONS.md` file with the universal commandments and project template (Appendix C); (ii) the Apptainer/Docker container definition used as the sandbox; (iii) the multi-node Slurm dispatcher (`remote-run`) and reporting helpers; (iv) a `README.md` describing setup (under a minute on a fresh system with `uv`). To reproduce the workflow on a new project, a researcher clones the repository, drops their existing project (codebase, LaTeX notes, references) into a directory, and launches a CLI agent (e.g., Claude Code) inside the sandbox; the agent reads `INSTRUCTIONS.md`, asks scoping questions, fills in the project-specific template, and begins autonomous execution.