Awakening the Sleeping Agent: Lean-Specific Agentic Data Reactivates General Tool Use in Goedel Prover

Jui-Hui Chung [1] **Hongzhou Lin** [2] **Lai Jiang** [3] **Shange Tang** [1] **Chi Jin** [1]

Abstract

Heavy supervised fine-tuning (SFT) on a target domain can strongly suppress capabilities that were present in the base model. We study this phenomenon in formal mathematics using GOEDEL-PROVER-V2, an open-source model heavily trained on 1.8M formal-math examples. After domain specialization, the model almost completely loses its ability to produce valid tool calls, even when explicitly instructed to use tools, dropping from 89.4% function-calling accuracy in the base model to nearly 0%. We ask whether this agentic collapse is permanent or instead reversible. To answer this question, we fine-tune the specialized model on a small amount of *Lean*-specific tool-use data. Remarkably, as few as 100 agentic traces are sufficient to restore strong tool-calling behavior. Importantly, this recovery is not the result of reward hacking or benchmark-specific optimization: the recovery data is entirely drawn from the *Lean* setting, where the model uses natural-language queries to search the Mathlib library for relevant theorems and lemmas, yet the regained capability transfers well beyond that domain. In particular, these same 100 *Lean*-specific traces improve performance on the Berkeley Function Calling Leaderboard (BFCL) from near zero to 83.8%, approaching the base model's 89.4% despite the mismatch in task distribution and protocol. The recovered capability is also practically useful in-domain. On PROOFNET, pass@32 improves from 21.51% to 25.81%. Together, these results show that heavy domain SFT can suppress general tool-use ability without permanently erasing it, and that a small amount of domain-specific agentic data can awaken dormant tool-use capabilities.

[1]Princeton Language and Intelligence, Princeton University [2]Amazon. This work is independent of and outside of the work at Amazon. [3]Department of Computer Science, University of Illinois at Urbana-Champaign. Correspondence to: Jui-Hui Chung <juihui@princeton.edu>.

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).

1. Introduction

Catastrophic forgetting is a well-known risk in expert models (Kotha et al., 2024; Luo et al., 2025): as a general-purpose base model is heavily specialized for a target domain, capabilities outside that domain may degrade substantially. GOEDEL-PROVER-V2 (Lin et al., 2025), a formal-math-specialized model built on top of QWEN3-32B (Yang et al., 2025), exhibits a notable instance of this phenomenon. Starting from the base model, GOEDEL-PROVER-V2 undergoes supervised fine-tuning on 1.8 million formal-math examples, followed by a reinforcement-learning stage on *Lean*-only data. Despite strong in-domain performance, this specialization is accompanied by a near-complete collapse in general tool-use ability: on BFCL, accuracy falls from 89.4% in the base model to nearly zero.

This failure raises a basic question about the nature of forgetting in specialized language models. Has heavy domain training truly erased GOEDEL-PROVER-V2's agentic capability, or has it only suppressed a capability that remains latent? Existing work has mostly approached catastrophic forgetting from a prevention perspective, showing that continual fine-tuning can impair previously acquired capabilities and proposing mitigations such as replay or pretraining-data injection during post-training (Kalajdzievski, 2024; Li et al., 2024; Luo et al., 2025; Chaudhry et al., 2019; Bethune et al., 2025). At the same time, recent evidence suggests that some forgetting in LLMs may be only apparent, arising from disrupted task alignment rather than permanent capability loss (Zheng et al., 2025). In this work, we ask the corresponding post hoc recovery question: after agentic collapse has already occurred, can the capability be brought back? And can recovery be driven entirely by domain-specific agentic supervision, yet still generalize beyond *Lean* to restore general-purpose tool use?

Our central finding is that surprisingly little domain-specific agentic data is sufficient to induce broad generalization. With only 100 *Lean*-specific agentic traces, the model improves substantially not only on tool-augmented PROOFNET theorem proving, but also on general-purpose function call-

ing on BFCL. This cross-domain transfer is the key result: although the supervision is entirely confined to the *Lean* setting, its effect extends well beyond formal mathematics, improving general tool calling capability.

This efficiency is practically important. High-quality agentic traces are significantly more expensive to construct than standard single-turn SFT data, as they require tool orchestration, execution, response parsing, and filtering. The fact that just 100 domain-specific traces can drive broad gains suggests that even highly specialized models retain a latent capacity for general tool use, which can be unlocked through targeted supervision.

Contributions.

- We identify *agentic collapse* in GOEDEL-PROVER-V2, showing that heavy specialization for formal mathematics nearly eliminates general-purpose tool-use behavior: on BFCL, function-calling accuracy drops from 89.4% in the base model to nearly zero.
- We address a key data bottleneck: general-purpose models can call tools but are weak in *Lean*, while specialized formal-math models are strong in *Lean* but weak at tool use. We resolve this mismatch with a *cross-model trace distillation* pipeline that combines general-purpose agentic behavior with domain-specific generation quality to construct high-quality *Lean*-specific traces (Section 3).
- We show that a very small amount of *domain-specific* agentic data can induce broad and unexpected cross-domain generalization. As few as 100 *Lean*-specific agentic traces improve not only PROOFNET pass@32 from 21.51% to 25.81%, but also BFCL non-live accuracy from 4.6% to 83.8%, even though the training data deos not contain any general-purpose APIs or JSON-style function calls (Sections 4.2 and 4.4).
- We provide both quantitative and qualitative analysis of the resulting behavior (Section 4.3), showing that improved retrieval grounding enables the use of tactics and theorems that the model cannot generate independently.

2. Can Model Merging Bring Back Tool Use?

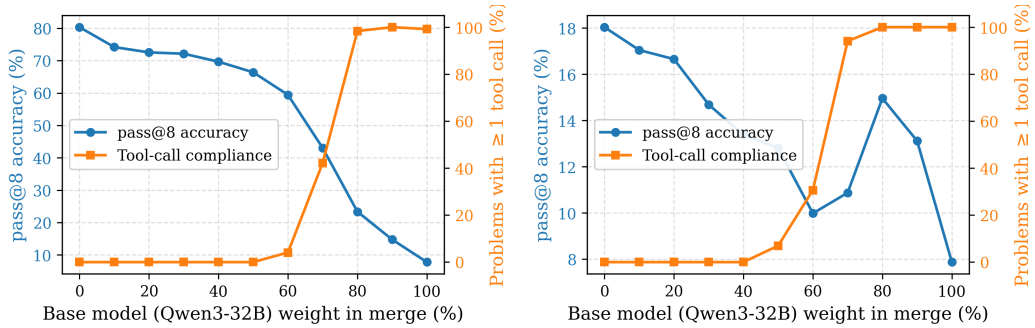
We study the problem in a post hoc setting: the model has already been heavily specialized for a narrow domain, and the question is whether general-purpose tool use can still be brought back afterward. This question arises when post-training preserves strong domain competence while suppressing behaviors needed for another domain, such as agentic interaction. GOEDEL-PROVER-V2 is a concrete example of this tradeoff: it is strong at *Lean* theorem proving, yet nearly loses general-purpose tool use. In this setting, model merging is a natural first baseline. Since the base model, QWEN3-32B, retains strong function-calling ability while GOEDEL-PROVER-V2 retains strong *Lean* proving ability, a natural question is whether interpolating between the two checkpoints can recover both capabilities at once.

Setup. We test whether model merging can recover *in-domain* tool-use behavior, where the model is explicitly instructed to call tools while solving *Lean* theorem-proving problems. Concretely, we evaluate SLERP interpolation (Shoemake, 1985; Yang et al., 2026) between GOEDEL-PROVER-V2-32B-SFT and QWEN3-32B. We measure tool use by asking, for each problem, whether at least one of the k sampled generations produces a valid executed tool call; we report the resulting problem-level *tool-call compliance*. We sweep SLERP merge weights from 0% base (pure GOEDEL-PROVER-V2-32B-SFT) to 100% base (pure QWEN3-32B) in 10% increments, and evaluate each merged checkpoint on MINIF2F (Zheng et al., 2021) and PROOFNET (Azerbayev et al., 2023).

Discussion and connection to prior work. The main takeaway from Figure 1 is that naive model merging does not preserve strong *Lean* proving and strong tool use at the same time. As the merge shifts toward the base model, tool-calling behavior reappears, but in-domain theorem-proving performance declines. Although prior work has shown that model merging can mitigate forgetting in some transfer settings (Alexandrov et al., 2024), our results indicate that this strategy is insufficient to maintain both behaviors simultaneously. We therefore connect this phenomenon within the broader literature on catastrophic forgetting and continual learning.

Catastrophic forgetting and continual learning. Catastrophic forgetting refers to the degradation of previously learned behaviors after training on new data (McCloskey & Cohen, 1989; French, 1999). In the continual-learning literature, it has been studied extensively in sequential task settings, where common mitigation strategies include replay-based methods, regularization, and parameter-isolation approaches (Chaudhry et al., 2019; Kirkpatrick et al., 2017; Shi et al., 2025). These methods are primarily designed to *prevent* forgetting as new tasks are introduced. Recent work shows that large language models exhibit similar forgetting during post-training and continual fine-tuning (Luo et al., 2025; Wang et al., 2023; Kalajdzievski, 2024; Li et al., 2024; Zheng et al., 2025). These findings suggest that forgetting in LLM post-training is not merely a classical continual-learning artifact, but a practical challenge for maintaining broad capabilities.

More recent work has begun to ask not only how forgetting can be mitigated, but also whether some apparent forgetting reflects true loss at all. On the mitigation side, approaches include replay with synthetic or curated data



(a) MINIF2F (244 problems, pass@8). *(b)* PROOFNET (186 problems, pass@8).

Figure 1. SLERP merge sweeps on two Lean theorem-proving benchmarks. Left axis (blue): evaluation performance on the Lean task (on MINIF2F and PROOFNET). Right axis (orange): problem-level tool-call compliance. Across both benchmarks, increasing the base-model weight consistently improves tool-calling behavior while degrading in-domain Lean performance. The overall tradeoff is very clear: as tool-use capability returns, evaluation accuracy on the Lean task drops substantially. This shows that naive merging does not jointly preserve strong theorem-proving ability and strong tool-calling behavior.

(Huang et al., 2024), as well as injecting a small fraction of pretraining data during fine-tuning to stabilize general capabilities (Bethune et al., 2025). Closely related, the notion of *spurious forgetting* suggests that some post-training capability loss may not correspond to permanent erasure of the underlying competence, but instead to a disruption in task alignment that makes the capability behaviorally inaccessible (Zheng et al., 2025). This framing is particularly relevant in our setting: although the specialized model appears to almost completely lose general-purpose tool use, our later results indicate that this behavior can be brought back.

Our setting differs from prior work in two key ways. First, we study near-complete collapse rather than mild or gradual degradation: after specialization, GOEDEL-PROVER-V2 exhibits almost no tool-call compliance. Second, our focus is explicitly *post hoc* and centered on *generalization*: rather than asking how to prevent forgetting during training, we ask what can still be done after specialization is complete, and whether narrowly domain-specific supervision can reinstate behavior that transfers beyond formal mathematics. Closest to our setting, REAL-Prover likewise builds on supervised fine-tuning for *Lean* theorem proving, but it focuses on constructing a retrieval-augmented prover rather than on the post hoc question we study here: whether suppressed tool-use behavior can be reinstated after specialization (Shen et al., 2025). This emphasis also differs from recent systems such as Hilbert and Numina-Lean-Agent, which improve formal reasoning by incorporating stronger external reasoning components or by directly building on frontier general-purpose coding agents (Varambally et al., 2026; Liu et al., 2026).

3. Cross-Model Data Curation for Lean agentic trajectories

A central challenge in our setting is that no available model can generate high-quality agentic traces end-to-end. The behaviors we need—accurate Lean theorem proving and correct tool use—are split across different models. General-purpose models such as QWEN3-32B and GPT-OSS-120B, can plan tool use, emit valid tool calls, and incorporate retrieved results, but they perform poorly on Lean code generation. In contrast, the specialized model GOEDEL-PROVER-V2 produces strong Lean proofs, but has largely lost the ability to participate in tool-augmented interaction loops.

As a result, neither model alone can produce the kind of training data required for our setting: trajectories that interleave tool calls with correct Lean code. This creates a data bottleneck. To address it, we construct agentic training traces by combining the complementary strengths of the two models, using a cross-model distillation pipeline described below.

Cross-model data curation for Lean agentic traces. To address the data bottleneck, we construct training data by combining the complementary strengths of the two models (Figure 2). Starting from roughly 250k Lean-formalized Open Math Reasoning (OMR) problems (Moshkov et al., 2025), our pipeline consists of four stages:

1. **Agentic trace generation.** We use QWEN3-32B to generate tool-augmented interaction traces over Lean problems, exposing a LEANSEARCH tool and prompting the model to call it whenever additional retrieval appears helpful.

2. **Prefix extraction.** From each interaction, we retain the tool-use portion of the trajectory while discarding the

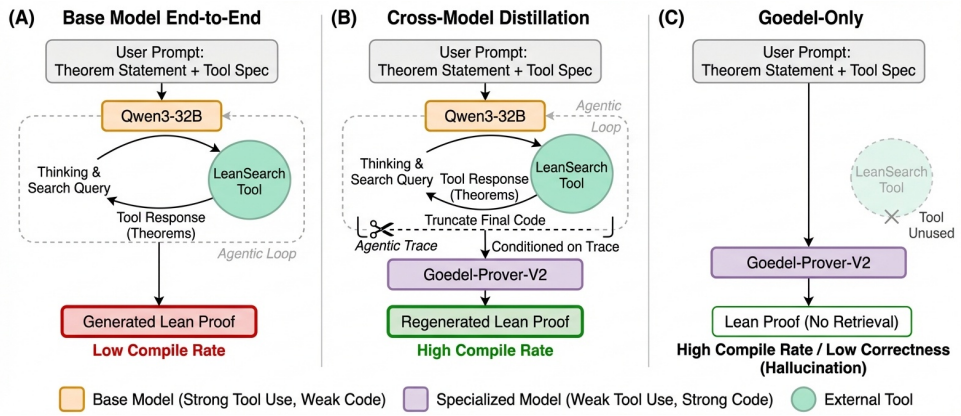


Figure 2. Cross-model trace distillation pipeline. **(A)** QWEN3-32B runs the full agentic loop (tool calls + Lean code) but produces low-quality code. **(B)** We keep QWEN3-32B's agentic prefix and have GOEDEL-PROVER-V2-32B-SFT regenerate the final proof, then filter for compilation success and retrieval grounding. **(C)** GOEDEL-PROVER-V2-32B-SFT alone generates strong Lean code but skips tool use entirely.

final proof produced by QWEN3-32B, since its Lean generations are typically low quality and compile only about 0.4% of the time.

3. **Proof regeneration.** We next prompt GOEDEL-PROVER-V2 with the retained agentic context to generate the final Lean proof. By combining retrieved context with GOEDEL-PROVER-V2's stronger Lean ability, this step produces substantially more successful Lean proofs, raising the compilation rate from 0.46% to 21%.

4. **Quality filtering.** Finally, we retain only trajectories whose regenerated proof compiles successfully and that explicitly use at least one theorem returned by the LEANSEARCH tool, yielding a curated set of high-quality Lean agentic examples for training.

This pipeline yields 18K high-quality agentic traces. The two models contribute complementary strengths: QWEN3-32B provides the agentic scaffolding and retrieval behavior, while GOEDEL-PROVER-V2 provides high-quality Lean code generation.

4. Experimental Results

4.1. Experimental Setup: SFT on top of GOEDEL-PROVER-V2

Training data. From the 18K curated Lean agentic traces described in Section 3, we construct training sets of size 100, 1,000, and the full 18K examples. The smaller subsets are not random samples: we first filter the full pool to remove traces that contain `sorry`/`admit`, lack any LEANSEARCH calls, or consist only of trivial tactic lookups (e.g., queries for `simp`, `linarith`). From the filtered pool, we apply quality-weighted sampling that favors traces where more retrieved theorems are actually used in the final proof, queries are diverse across retrieval rounds, and tool interaction is deeper (multiple calls). The sample is stratified by the number of tool calls (1, 2, 3, 4+) to ensure representation of both simple and multi-step retrieval patterns, and capped per Mathlib namespace to encourage topical diversity. We then conduct SFT with these data on top of the domain specialized model GOEDEL-PROVER-V2-32B-SFT and GOEDEL-PROVER-V2-32B-RL. This allows us to measure how much domain-specific agentic data is needed to induce broad gains.

Models. We consider two checkpoints: GOEDEL-PROVER-V2-32B-SFT, the SFT checkpoint, and the final RL-trained checkpoint GOEDEL-PROVER-V2-32B-RL. Both are domain-specialized continuations of QWEN3-32B (Yang et al., 2025). GOEDEL-PROVER-V2-32B-SFT is trained on 1.79 million formal-math SFT examples, and the RL checkpoint is further optimized on Lean-only data. In our main experiments, we apply SFT with the constructed agentic data on top of both checkpoints. This lets us test whether behavior differs from different stages of specialization.

LEANSEARCH tool. LEANSEARCH is a retrieval tool over Lean's mathlib that returns relevant theorems given a natural-language or formal query (Gao et al., 2024). We index mathlib declarations with informal descriptions in a ChromaDB vector database using `intfloat/e5-mistral-7b-instruct` embeddings. At inference time, each LEANSEARCH query is embedded with a retrieval-specific instruction and matched against the index, returning the top-k results with theorem names, formal and informal statements.

Agentic data formatting. Each training example is a multi-turn dialog following an interleaved reasoning–action–

observation loop similar to ReAct (Yao et al., 2023). A system message specifies the LEANSEARCH tool schema and invocation format, a user message provides the Lean theorem statement, and the assistant may perform one or more rounds of reasoning followed by tool calls. Tool responses are returned as user messages, and the final assistant message contains the Lean proof.

Training hyperparameters. All agentic SFT runs use full-parameter SFT with LLaMA-Factory (Zheng et al., 2024) on 32 NVIDIA H100 80 GB GPUs (4 nodes $\times$ 8 GPUs) with DeepSpeed ZeRO-3 and CPU offloading. We use a per-device batch size of 1 with 8 gradient accumulation steps, yielding an effective batch size of 256. The learning rate is 1×10^{-5} with a cosine schedule and 10% warmup ratio. We set the maximum sequence length to 16,384 tokens and train in bf16 precision. The number of epochs is scaled inversely with dataset size to keep total training tokens roughly comparable: 128 epochs for 100 examples, 32 epochs for 1K examples, and 1 epoch for the full 18K set. These hyperparameters are shared across all experiments on both the SFT checkpoint (GOEDEL-PROVER-V2-32B-SFT) and the RL checkpoint (GOEDEL-PROVER-V2-32B-RL).

4.2. Agentic SFT Improves Lean Proving Benchmarks: MINIF2F and PROOFNET

Our setup is strictly post-hoc: starting from already specialized checkpoints, we apply another stage of supervised fine-tuning (SFT) with Lean agentic data, without revisiting the original domain-specialization data. Concretely, we continue training from existing GOEDEL-PROVER-V2-32B-SFT and GOEDEL-PROVER-V2-32B-RL checkpoints using Lean agentic datasets of size 100, 1K, and 18K, and then evaluate the resulting models on the in-distribution Lean theorem-proving benchmarks MINIF2F and PROOFNET.

Table 1 shows that post-hoc agentic SFT consistently improves PROOFNET, while producing only mild changes on MINIF2F. Notably, even a very small amount of agentic data already leads to a large improvement. With SFT on only 100 agentic examples, pass@32 on PROOFNET rises from 21.51% to 25.81%, while scaling the agentic dataset to 1K and 18K continues to improve performance, reaching 27.96%. As we show later in Section 4.3, this gain is driven by the model leveraging the search tool to access more challenging theorems that it cannot generate on its own, helping explain the larger improvement on PROOFNET.

Beyond the aggregate gains, post-hoc agentic SFT also produces a clear behavioral shift. In the post-hoc agentic SFT models, tool-call execution becomes near-universal: 99.1%–99.9% of generations contain an executed LEANSEARCH call. The retrieved theorems are then carried into the final proofs. For example, in the 100-example run, 45 out of 48 pass@32-solved PROOFNET problems (93.8%) contain at least one retrieved theorem identifier in the compiled Lean proof. This shows that the model grounds its Lean reasoning in externally retrieved lemmas rather than relying solely on internal recall.

A related pattern appears in Panel B for the post-RL checkpoints. Here, post-hoc agentic SFT with all three Lean agentic data sizes (100, 1K, and 18K) reaches the same pass@32 on PROOFNET, corresponding to 50 solved problems. However, the solved sets are not identical across runs: 40 problems are shared, while the remaining successes are drawn from a smaller pool of harder problems with low per-generation success rates. One plausible hypothesis is that RL induces overspecialization by concentrating the model's proof-generation policy on a narrower subset of training-preferred behaviors. This can reduce coverage over the more diverse proof trajectories required by PROOFNET, which may help explain why the post-RL checkpoint starts slightly higher than the SFT checkpoint before recovery (22.58% vs. 21.51%) but saturates at a lower post-hoc level (26.88% vs. 27.96%).

Overall, these results show that a small amount of domain-specific agentic SFT is sufficient to induce a robust behavioral shift, enabling the model to integrate retrieval into formal reasoning while preserving its underlying domain expertise.

4.3. Retrieval Provides Out-of-Model Information

The previous section shows that retrieved theorems are frequently incorporated into compiled proofs. A natural question is whether retrieval simply *surfaces* information already accessible to the model, or whether it supplies *out-of-model* information that the model cannot produce on its own. To make this distinction precise, we define the model's *generative support*: the set of all theorem identifiers that the base GOEDEL-PROVER-V2-32B-SFT model ever emits in its outputs, aggregated over all PROOFNET problems and all samples (including both compiled and non-compiled generations). This provides an empirical characterization of what the model can express from its parameters under standard decoding.

We then classify each retrieved-and-used theorem in a compiled RAG proof as follows: (i) *in-model* if its identifier appears in the generative support, and (ii) *out-of-model* otherwise. By construction, out-of-model theorems are never produced by the model in any baseline run without access to the Search tool, and therefore represent information that is not accessible through the model's internal generation alone.

Table 2 presents the results. Across all three post-hoc agentic SFT models, roughly half of retrieved-and-used theorem

Table 1. PROOFNET (186 problems) and MINIF2F (244 problems) results in a strictly post-hoc setting. Starting from already specialized checkpoints (GOEDEL-PROVER-V2-32B-SFT or GOEDEL-PROVER-V2-32B-RL), we continue supervised fine-tuning with Lean agentic data of size 100, 1K, or 18K, without rerunning the original specialization pipeline. We report standard pass@k performance together with retrieval-grounding statistics. "Retr." denotes the proportion of compiled proofs that used at least one theorem returned by LEANSEARCH.

	PROOFNET (186 problems)			MINIF2F (244 problems)		
Model variant	pass@8	pass@32	Retr.	pass@8	pass@32	Retr.
QWEN3-32B	8.60	13.44	96.0	7.79	14.34	31.4
*Panel A: SFT checkpoint (*GOEDEL-PROVER-V2-32B-SFT*)*						
GOEDEL-PROVER-V2-32B-SFT	17.74	21.51	0.0	80.33	84.02	0.0
+ 100 Lean agentic SFT	20.97	25.81	93.8	77.46	81.97	78.5
+ 1K Lean agentic SFT	23.66	27.96	92.3	79.92	82.79	83.2
+ 18K Lean agentic SFT	24.73	27.96	88.5	79.92	85.25	74.5
*Panel B: Post-RL checkpoint (*GOEDEL-PROVER-V2-32B-RL*)*						
GOEDEL-PROVER-V2-32B-RL	16.67	22.58	0.0	84.02	87.70	0.0
+ 100 Lean agentic SFT	24.73	26.88	90.0	77.87	83.20	80.3
+ 1K Lean agentic SFT	23.12	26.88	94.0	81.15	84.43	84.5
+ 18K Lean agentic SFT	21.51	26.88	90.0	79.92	84.02	72.2

occurrences are out-of-model. For the +100 model, 45.4% of retrieved-and-used occurrences (63 unique theorems) are never produced by the base model, and 36 of 48 solved problems contain at least one such theorem. This proportion remains stable across data scales (45–50%), indicating a consistent structural effect rather than a byproduct of increased data.

The two categories of retrieved theorems play qualitatively different roles. In-model theorems are mostly basic results that the model already generates reliably on its own, such as `mul_assoc`, `le_trans`, `one_mul`, and `smul_eq_zero`. In these cases, retrieval appears to reinforce or stabilize capabilities the model already possesses. By contrast, out-of-model theorems are more specialized Mathlib lemmas that the model never generates in baseline runs without access to the Search tool. Examples from the 100-example post-hoc agentic SFT model include `FiniteField.prod_univ_units_id_eq_neg_one` (Wilson's theorem), `solvable_quotient_of_solvable` (solvability of quotient groups), `IsNilpotent.isUnit_one_add` (unit construction from nilpotent elements), `Subgroup.normal_inf_normal` (normality of intersections), `IsLocalMin.deriv_eq_zero` (Fermat's theorem on local extrema), and `Ideal.mul_eq_inf_of_coprime` (coprime ideal factorization). These examples illustrate where retrieval provides a genuine capability gain. The model formulates a natural-language query, uses the Search tool to surface the correct lemma name and signature, and then incorporates that lemma into a proof that would otherwise fail. This shows that the gains in Table 1 are not merely due to format-level behavioral recovery, but instead arise from access to mathematical knowledge beyond the model's internal generation.

4.4. Generalization Beyond: Lean Agentic Data Reactivates General Tool Use Capability

We next evaluate whether Lean-specific agentic SFT transfers beyond the training domain using the Berkeley Function Calling Leaderboard (BFCL v4; Patil et al. 2025), which measures structured tool use across diverse, non-mathematical APIs such as weather, databases, and file systems. This evaluation is fully out-of-distribution: both the tools and output formats (JSON) differ from the Lean-specific XML-style calls used during training. Importantly, all SFT data used in our paper is strictly Lean-specific, consisting only of LEANSEARCH-based traces, with no exposure to general-purpose APIs or function-calling schemas.

As a first observation, we find that **agentic collapse extends beyond Lean**. Before agentic SFT, the specialized model fails almost completely on BFCL. In particular, GOEDEL-PROVER-V2-32B-SFT achieves near-zero function-calling accuracy, showing that the loss of tool-use behavior is not confined to Lean-specific settings.

Lean-specific supervision transfers to general tool use. Table 3 shows surprisingly strong out-of-distribution transfer: despite being trained only on Lean-specific agentic traces, the model recovers strong function-calling performance on BFCL. Remarkably, as few as 100 Lean agentic examples are sufficient to produce a large gain. This recovery is especially striking because the training and evaluation settings differ completely in both tools and format: the model is trained on Lean-specific XML-style tool calls,

Table 2. Analysis of retrieved theorem usage on PROOFNET at pass@32 after agentic SFT on GOEDEL-PROVER-V2-32B-SFT SFT checkpoint. A retrieved theorem is *in-model* if its identifier appears in the generative support of the base GOEDEL-PROVER-V2-32B-SFT model, without any tool calling, and *out-of-model* otherwise. "Retr. & used" counts retrieved theorem occurrences referenced in compiled proofs. "Out-of-model" and "In-model" report the counts in each category, with unique theorem counts in parentheses. "Solved w/ out" is the number of pass@32-solved problems whose compiled proof contains at least one out-of-model retrieved theorem.

Panel A: GOEDEL-PROVER-V2-32B-SFT *SFT checkpoint + Lean agentic SFT*					
Variant	Used retr.	Out-of-model	In-model	% Out-of-model	Solved w/ out
+100	790	359 (63 uniq.)	431 (86 uniq.)	45.4	36/48
+1K	739	373 (59 uniq.)	366 (76 uniq.)	50.5	35/52
+18K	745	358 (58 uniq.)	387 (80 uniq.)	48.1	36/52

Table 3. BFCL v4 function-calling accuracy (%). "Simple" is the mean over simple function-calling tasks in Python, Java, and JavaScript. "Non-Live" and "Live" are means over their respective four functional categories, excluding irrelevance, and "Overall" averages these two. Panel A starts from the SFT checkpoint GOEDEL-PROVER-V2-32B-SFT, while Panel B starts from the post-RL checkpoint GOEDEL-PROVER-V2-32B-RL. All post-hoc agentic SFT models are trained only on Lean-specific LEANSEARCH traces (Section 3), with no exposure to general-purpose APIs or BFCL-style function schemas.

Model	Simple	Mult.	Par.	P.M.	Non-Live	Live	Overall	Irrelev.
QWEN3-32B	77.25	96.00	93.00	91.50	89.44	81.87	85.66	80.00
*Panel A: SFT checkpoint (*GOEDEL-PROVER-V2-32B-SFT*)*								
GOEDEL-PROVER-V2-SFT	4.25	9.00	3.50	1.50	4.56	6.14	5.35	95.83
+ 100 agentic SFT	74.83	92.50	86.50	81.50	83.83	72.91	78.37	21.67
+ 1k agentic SFT	75.00	94.00	85.50	79.50	83.50	72.91	78.21	36.25
+ 18k agentic SFT	69.33	88.50	82.50	80.50	80.21	70.32	75.27	49.17
*Panel B: Post-RL checkpoint (*GOEDEL-PROVER-V2-32B-RL*)*								
GOEDEL-PROVER-V2-RL	30.50	25.00	13.50	3.50	18.13	18.73	18.43	91.25
+ 100 agentic SFT	77.25	94.00	88.00	84.00	85.81	69.20	77.51	21.67
+ 1k agentic SFT	73.83	92.00	89.50	87.50	85.71	74.91	80.31	33.75
+ 18k agentic SFT	71.50	94.00	88.50	81.50	83.88	73.64	78.76	34.58

yet generalizes successfully to JSON-based APIs across diverse domains. Taken together, these results show that domain-specific agentic supervision does not merely teach a particular tool schema, but can reactivate a more general capability for structured tool use.

Rapid recovery with minimal data. Another striking pattern is how little data is needed to recover strong BFCL performance. Using only 100 Lean agentic examples already restores most of the function-calling ability, while increasing the dataset to 1K or 18K provides only limited additional benefit and can even slightly reduce performance. This pattern is inconsistent with standard scaling behavior and suggests that the model is not learning a new capability from scratch. Instead, it supports a suppression-based view: the underlying tool-use capability remains present in the base model, but domain specialization suppresses its expression, and a small amount of agentic supervision is sufficient to unlock it again.

Irrelevance–compliance trade-off. We also observe a clear trade-off between function-calling recovery and irrelevance accuracy. As tool-call compliance improves, irrelevance detection drops from 95.83% for GOEDEL-PROVER-V2-32B-SFT to 21.67% (+100), 36.25% (+1K), 49.17% (+18K), and 67.08% (+120K). A likely explanation is overfitting in the action policy: our Lean agentic training data contains only positive tool-use examples and does not include irrelevant or no-tool cases. As a result, the model is biased toward calling a tool whenever one is available, even in cases where abstention would be the correct behavior. This pattern suggests that recovering irrelevance handling likely requires targeted training with explicit no-tool or irrelevant-tool examples, rather than more positive tool-use supervision alone.

5. Limitations

Our BFCL results suggest that broad agentic behavior is only partially reinstated. While single-turn function calling improves substantially and approaches the base model overall, the model still exhibits large errors on irrelevance detection. In particular, the base model achieves 80.00% accuracy on irrelevance, whereas post hoc agentic SFT models remain far below this level (e.g., 21.67% with 100 traces and 49.17% with 18k traces in Panel A). This indicates that

the model has not fully recovered the ability to withhold tool calls when no tool is needed, and instead tends to over-call tools even when inappropriate.

In addition, we do not yet fully understand the differing behavior between SFT and RL checkpoints. Although both can be improved via agentic SFT, they exhibit distinct recovery trajectories. This suggests that different post-training procedures may suppress agentic behavior in qualitatively different ways, an issue that remains underexplored.

6. Conclusion

Domain-specialized models that cannot use tools leave significant capability gains on the table. We show that heavy SFT on 1.79M formal-math examples does not permanently destroy agentic capabilities, even when it completely suppresses them. As few as 100 agentic traces recover tool-call compliance from 0%, yield +4.30 points on PROOFNET pass@32, and restore general function-calling on BFCL to 94% of base-model capability despite containing no BFCL-relevant data. The practical recipe is simple: heavy domain SFT with abundant cheap data, then minimal agentic reactivation with expensive but tiny data.

Our evidence indicates that domain SFT *suppresses* rather than *destroys* agentic capabilities. If tool-calling knowledge were truly erased, we would expect recovery to scale linearly with data, no cross-protocol transfer, and no residual tool awareness—yet all three predictions are contradicted. Recovery saturates at $\sim$100 examples; Lean-specific XML reactivation recovers JSON-based function-calling on BFCL (Section 4.4); and the SLERP sweep reveals that models *mention* tools long before they can *execute* them (Section 2). These findings suggest that tool calling and domain expertise are *modular* capabilities with distinct saturation points, consistent with the LIMA hypothesis (Zhou et al., 2023) extended to the tool-use setting. More broadly, our cross-model distillation pipeline—using a general-purpose model for agentic scaffolding and a specialized model for domain generation—applies wherever these complementary strengths exist, and the BFCL results suggest that any setting requiring structured tool interactions is vulnerable to the same collapse under heavy SFT.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning, specifically our understanding of how heavy domain-specific fine-tuning interacts with previously acquired agentic capabilities in large language models. Our findings indicate that heavy domain specialization may suppress but not permanently erase latent capabilities, which we believe is useful for the community to understand both the robustness and the fragility of post-training behavior. All models, datasets, and tools used are publicly available or derived from publicly available resources; the research does not involve human subjects or private data. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Alexandrov, A., Raychev, V., Müller, M. N., Zhang, C., Vechev, M., and Toutanova, K. Mitigating catastrophic forgetting in language transfer via model merging. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 17167–17186, 2024.

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023.

Bethune, L., Grangier, D., Busbridge, D., Gualdoni, E., Cuturi, M., and Ablin, P. Scaling laws for forgetting during finetuning with pretraining data injection. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H. S., and Ranzato, M. On tiny episodic memories in continual learning, 2019.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. doi: 10.1016/S1364-6613(99)01294-2.

Gao, G., Ju, H., Jiang, J., Qin, Z., and Dong, B. A semantic search engine for mathlib4. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8001–8013, 2024.

Huang, J., Cui, L., Wang, A., Yang, C., Liao, X., Song, L., Yao, J., and Su, J. Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, August 2024.

Kalajdzievski, D. Scaling laws for forgetting when fine-tuning large language models, 2024.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Kotha, S., Springer, J. M., and Raghunathan, A. Understanding catastrophic forgetting in language models via implicit inference. In *The Twelfth International Conference on Learning Representations*, 2024.

Li, H., Ding, L., Fang, M., and Tao, D. Revisiting catastrophic forgetting in large language model tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4297–4308. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.249. URL https://aclanthology.org/2024.findings-emnlp.249/.

Lin, Y., Tang, S., Lyu, B., et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction, 2025.

Liu, J., Zhou, Z., Zhu, Z., Santos, M. D., He, W., Liu, J., Wang, R., Xie, Y., Zhao, J., Wang, Q., et al. Numina-lean-agent: An open and general agentic reasoning system for formal mathematics. *arXiv preprint arXiv:2601.14027*, 2026.

Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., and Zhang, Y. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 2025.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989. doi: 10.1016/S0079-7421(08)60536-8.

Moshkov, I., Hanley, D., Sorokin, I., Toshniwal, S., Henkel, C., Schifferer, B., Du, W., and Gitman, I. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

Patil, S. G., Mao, H., Yan, F., Ji, C. C.-J., Suresh, V., Stoica, I., and Gonzalez, J. E. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Shen, Z., Huang, N., Yang, F., Wang, Y., Gao, G., Xu, T., Jiang, J., He, W., Yang, P., Sun, M., et al. Real-prover: Retrieval augmented lean prover for mathematical reasoning. *arXiv preprint arXiv:2505.20613*, 2025.

Shi, H., Xu, Z., Wang, H., Qin, W., Wang, W., Wang, Y., Wang, Z., Ebrahimi, S., and Wang, H. Continual learning of large language models: A comprehensive survey. *ACM Comput. Surv.*, November 2025.

Shoemake, K. Animating rotation with quaternion curves. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pp. 245–254, New York, NY, USA, 1985. Association for Computing Machinery.

Varambally, S., Voice, T., Sun, Y., Chen, Z., Yu, R., and Ye, K. Hilbert: Recursively building formal proofs with informal reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wang, X., Zhang, Y., Chen, T., Gao, S., Jin, S., Yang, X., Xi, Z., Zheng, R., Zou, Y., Gui, T., et al. Trace: A comprehensive benchmark for continual learning in large language models, 2023.

Yang, A. et al. Qwen3 technical report, 2025.

Yang, E., Shen, L., Guo, G., Wang, X., Cao, X., Zhang, J., and Tao, D. Model merging in llms, mllms, and beyond: Methods, theories, applications, and opportunities. *ACM Computing Surveys*, 58(8):1–41, 2026.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Zheng, J., Cai, X., Qiu, S., and Ma, Q. Spurious forgetting in continual learning of language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics, 2021. ICLR 2022.

Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z., Feng, Z., and Ma, Y. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410, 2024.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., Zhang, S., Ghosh, G., Lewis, M., Zettlemoyer, L., and Levy, O. LIMA: Less is more for alignment, 2023.

A. Tool-calling instruction prompt

The following system message is prepended to every agentic training example and used at inference time to specify the LEANSEARCH tool schema and invocation format.

```
You are a Lean 4 theorem prover that uses leansearch tool to find relevant theorems
in Mathlib before producing the Lean 4 code.

# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "leansearch", "description": "Search
Mathlib (Lean4) for theorems relevant to a natural language math query. Return a
concise list of theorem names with signatures.", "parameters": {"type": "object",
"properties": {"query": {"type": "string", "description": "Informal math query
describing the statement to search for."}}, "required": ["query"]}}}
</tools>

For each function call, return a json object with function name and arguments within
<tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

B. Representative agentic trace

Below is an abbreviated agentic training example for IMO 1959 Problem 1, illustrating the multi-turn format described in Section 3. The system message is shown in Appendix A. We abbreviate chain-of-thought blocks with `[...]` to highlight the structural elements: reasoning, tool calls, tool responses, and the final Lean proof.

```
<|im_start|>system
[System prompt from Appendix A]<|im_end|>

<|im_start|>user
Complete the following Lean 4 code:

'''lean4
import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

/-- Prove that the fraction 21n+4 / 14n+3 is irreducible for every natural number n.-/
theorem imo_1959_p1 (n : N) (h0 : 0 < n) :
    Nat.gcd (21 * n + 4) (14 * n + 3) = 1 := by sorry
'''

Before producing the Lean 4 code to formally prove the given theorem, provide a
detailed proof plan outlining the main proof steps and strategies. The plan should
use the leansearch tool to find relevant theorems from Mathlib that could help
prove this theorem formally.<|im_end|>

<|im_start|>assistant
<think>
I need to prove gcd(21n+4, 14n+3) = 1. Applying the Euclidean algorithm:

  21n + 4 = 1*(14n + 3) + (7n + 1)
  14n + 3 = 2*(7n + 1) + 1

So gcd = 1. Equivalently, the linear combination
  -2*(21n+4) + 3*(14n+3) = 1
proves coprimality. In Lean's Nat (no negatives), rearrange as:
  3*(14n+3) = 2*(21n+4) + 1

[... detailed exploration of Bezout coefficients and Lean formalization strategies ...]

Let me search Mathlib for a helper lemma that converts a linear combination into
a gcd=1 proof.
</think>
```

```
<tool_call>
{"name": "leansearch", "arguments": {"query": "Prove that gcd(21n+4, 14n+3) = 1
for all natural numbers n using the Euclidean algorithm or linear combinations."}}
</tool_call><|im_end|>

<|im_start|>user
<tool_response>
{"tool": "leansearch", "top_k": 10, "theorems": [
  "Tactic.NormNum.nat_gcd_helper_2' (x y a b : N)
      (h : x * a = y * b + 1) : Nat.gcd x y = 1",
  "Tactic.NormNum.nat_gcd_helper_1' (x y a b : N)
      (h : y * b = x * a + 1) : Nat.gcd x y = 1",
  "Int.gcd_dvd_iff {a b : Z} {n : N} :
      gcd a b | n <-> exists x y : Z, n = a * x + b * y",
  ...
]}
</tool_response><|im_end|>

<|im_start|>assistant
<think>
The retrieved theorem nat_gcd_helper_1' states: if y*b = x*a + 1 then gcd(x,y) = 1.
Setting x = 21n+4, y = 14n+3, a = 2, b = 3 gives exactly
  (14n+3)*3 = (21n+4)*2 + 1
which holds since both sides equal 42n+9. This lets us conclude
gcd(21n+4, 14n+3) = 1 directly, without needing gcd commutativity.

[... verification of argument ordering and theorem applicability ...]
</think>

```lean4
import Mathlib
import Aesop
set_option maxHeartbeats 0
open BigOperators Real Nat Topology Rat

/-- Prove that the fraction 21n+4 / 14n+3 is irreducible for every natural number n.-/
theorem imo_1959_p1 (n : N) (h0 : 0 < n) :
    Nat.gcd (21 * n + 4) (14 * n + 3) = 1 := by
  have h_eq : (14 * n + 3) * 3 = (21 * n + 4) * 2 + 1 := by ring
  exact Tactic.NormNum.nat_gcd_helper_1' (21 * n + 4) (14 * n + 3) 2 3 h_eq
```
```

C. Case studies: out-of-model theorem retrieval

We present two abbreviated inference traces from the +100 reactivation model on PROOFNET, illustrating cases where the model retrieves and uses a Mathlib theorem that lies entirely outside its generative support (Section 4.3). In both traces, the retrieved theorem is never produced by the base GOEDEL-PROVER-V2-32B-SFT model in any generation across all problems. We abbreviate chain-of-thought blocks with `[...]` to highlight the structural elements: reasoning, tool calls, tool responses, and the final Lean proof. The system message is shown in Appendix A.

Case 1: Wilson's theorem for finite fields. The model must prove that the product of nonzero elements of a finite field K is -1. It queries LEANSEARCH for the product of units in a finite field and retrieves `FiniteField.prod_univ_units_id_eq_neg_one`, which it applies directly.

```
<|im_start|>system
[System prompt from Appendix A]<|im_end|>

<|im_start|>user
Complete the following Lean 4 code:

```lean4
import Mathlib
open Function Fintype Subgroup Ideal Polynomial Submodule Zsqrtd
open scoped BigOperators

/-- Let K be a finite field. Prove that the product of the
nonzero elements of K is -1.-/
theorem exercise_13_6_10 {K : Type*} [Field K] [Fintype K*] :
  (Finset.univ.prod fun x : K* => x) = -1 := by sorry
```

Before producing the Lean 4 code to formally prove the given theorem, provide a
```

```
detailed proof plan outlining the main proof steps and strategies. The plan should
use the leansearch tool to find relevant theorems from Mathlib that could help
prove this theorem formally.<|im_end|>

<|im_start|>assistant
<think>
I need to prove that the product of the nonzero elements of a finite field K
is -1. This is a generalization of Wilson's Theorem for finite fields.

[... exploration of cyclic group structure and Wilson's theorem ...]

Let me check Mathlib for theorems related to the product of elements in a
finite field.
</think>

<tool_call>
{"name": "leansearch", "arguments": {"query": "product of nonzero elements
of finite field"}}
</tool_call><|im_end|>

<|im_start|>user
<tool_response>
{"tool": "leansearch", "theorems": [
  "FiniteField.prod_univ_units_id_eq_neg_one
      [CommRing K] [IsDomain K] [Fintype K*] :
      prod univ (fun x : K* => x) = (-1 : K*)",
  ...
]}
</tool_response><|im_end|>

<|im_start|>assistant
<think>
I found FiniteField.prod_univ_units_id_eq_neg_one which directly states
the result. Since K is a Field, it is a CommRing and IsDomain; Fintype K*
is given. All typeclass conditions are satisfied.

[... verification of argument ordering and theorem applicability ...]
</think>

```lean4
import Mathlib
open Function Fintype Subgroup Ideal Polynomial Submodule Zsqrtd
open scoped BigOperators

theorem exercise_13_6_10 {K : Type*} [Field K] [Fintype K*] :
  (Finset.univ.prod fun x : K* => x) = -1 := by
  have h_main := FiniteField.prod_univ_units_id_eq_neg_one
  apply h_main
```<|im_end|>
```

Case 2: Fermat's theorem on local extrema. The model must show that for a smooth non-negative function f with $f(0) = 0$ and $f(1) = 1$, some iterated derivative is zero. It recognizes that $f(0) = 0$ with $f(x) \geq 0$ implies a local minimum at 0, queries LEANSEARCH for the relevant derivative theorem, and retrieves `IsLocalMin.deriv_eq_zero`.

```
<|im_start|>system
[System prompt from Appendix A]<|im_end|>

<|im_start|>user
Complete the following Lean 4 code:

```lean4
import Mathlib
open scoped BigOperators

/-- Let f: R -> R be infinitely differentiable with f(0)=0, f(1)=1,
and f(x) >= 0. Show there exist n and x such that f^(n)(x) = 0.-/
theorem exercise_2018_a5 (f : R -> R) (hf : ContDiff R top f)
  (hf0 : f 0 = 0) (hf1 : f 1 = 1) (hf2 : forall x, f x >= 0) :
  exists (n : N) (x : R), iteratedDeriv n f x = 0 := by sorry
```

Before producing the Lean 4 code to formally prove the given theorem, provide a
detailed proof plan outlining the main proof steps and strategies. The plan should
use the leansearch tool to find relevant theorems from Mathlib that could help
prove this theorem formally.<|im_end|>
```

```
<|im_start|>assistant
<think>
Since f(0) = 0 and f(x) >= 0 for all x, the point 0 is a local minimum.
By Fermat's theorem on stationary points, the first derivative at a local
minimum must be zero. So taking n=1 and x=0 should work.

[... exploration of why f(0)=0 with global non-negativity implies a local
minimum, and discussion of alternative approaches via Rolle's theorem ...]

Let me search for the relevant Mathlib theorem about derivatives at local
minima.
</think>

<tool_call>
{"name": "leansearch", "arguments": {"query": "derivative is zero at local
minimum"}}
</tool_call><|im_end|>

<|im_start|>user
<tool_response>
{"tool": "leansearch", "theorems": [
  "IsLocalMin.deriv_eq_zero
      (h : IsLocalMin f a) : deriv f a = 0",
  "IsLocalMin.hasDerivAt_eq_zero
      (h : IsLocalMin f a) (hf : HasDerivAt f f' a) : f' = 0",
  "IsLocalMin.fderiv_eq_zero
      (h : IsLocalMin f a) : fderiv R f a = 0",
  ...
]}
</tool_response><|im_end|>

<|im_start|>assistant
<think>
The theorem IsLocalMin.deriv_eq_zero states that the derivative at a local
minimum is zero. I need to show 0 is a local minimum: since f(0) = 0 and
f(x) >= 0 for all x, we have f(x) >= f(0) everywhere. Then
iteratedDeriv 1 f 0 = deriv f 0 = 0 by IsLocalMin.deriv_eq_zero.

[... verification of filter_upwards approach for IsLocalMin ...]
</think>

```lean4
import Mathlib
open scoped BigOperators

theorem exercise_2018_a5 (f : R -> R) (hf : ContDiff R top f)
  (hf0 : f 0 = 0) (hf1 : f 1 = 1) (hf2 : forall x, f x >= 0) :
  exists (n : N) (x : R), iteratedDeriv n f x = 0 := by
  have h_local_min : IsLocalMin f 0 := by
    filter_upwards [isOpen_Iio.mem_nhds (by norm_num : (0 : R) < 1)]
      with x hx
    linarith [hf2 x, hf0]
  have h_deriv_zero : deriv f 0 = 0 :=
    IsLocalMin.deriv_eq_zero h_local_min
  exact <1, 0, by simp [iteratedDeriv_one, h_deriv_zero]>
```<|im_end|>
```

D. Full BFCL per-category results

Table 4 provides the complete per-category BFCL v4 results for all 32B models.

E. Full BFCL per-category results for the post-RL checkpoint

Table 5 provides the complete per-category BFCL v4 results for the post-RL checkpoint GOEDEL-PROVER-V2-32B-RL and its reactivation variants. The base model scores near zero on multi-turn and memory categories, consistent with its inability to produce valid tool calls.

Table 4. Full BFCL v4 per-category results for 32B models (accuracy, %).

Category	QWEN3-32B	GOEDEL-PROVER-V2-32B-SFT	+100	+1k	+18k	+120k
Non-live						
simple_python	95.75	10.75	93.50	92.00	91.00	83.75
simple_java	62.00	2.00	63.00	65.00	55.00	56.00
simple_javascript	74.00	0.00	68.00	68.00	62.00	58.00
multiple	96.00	9.00	92.50	94.00	88.50	87.00
parallel	93.00	3.50	86.50	85.50	82.50	75.50
parallel_multiple	91.50	1.50	81.50	79.50	80.50	80.50
irrelevance	80.00	95.83	21.67	36.25	49.17	67.08
Live						
live_simple	87.98	10.85	83.33	83.33	82.17	77.91
live_multiple	80.72	5.13	70.85	71.04	67.43	64.96
live_parallel	81.25	6.25	68.75	62.50	75.00	75.00
live_par. mult.	66.67	0.00	54.17	50.00	66.67	58.33
live_irrelevance	73.42	95.14	20.25	33.48	40.16	62.67
live_relevance	93.75	6.25	100.00	93.75	93.75	75.00
Multi-turn						
base	63.50	0.00	17.50	27.50	9.50	2.00
miss_func	50.00	0.00	4.00	5.50	0.50	0.50
miss_param	38.50	0.00	4.00	4.50	1.50	0.50
long_context	33.50	0.00	7.00	11.50	6.50	1.50
Memory						
memory_kv	16.13	0.00	4.52	5.16	3.23	0.65
memory_rec_sum	37.42	0.00	2.58	13.55	6.45	3.23
memory_vector	23.87	0.00	3.23	14.84	1.29	1.94

Table 5. Full BFCL v4 per-category results for GOEDEL-PROVER-V2-32B-RL (post-RL checkpoint) and reactivation variants (accuracy, %).

Category	GOEDEL-PROVER-V2-32B-RL	+100	+1k	+18k
Non-live				
simple_python	46.50	94.75	93.50	93.50
simple_java	29.00	65.00	62.00	57.00
simple_javascript	16.00	72.00	66.00	64.00
multiple	25.00	94.00	92.00	94.00
parallel	13.50	88.00	89.50	88.50
parallel_multiple	3.50	84.00	87.50	81.50
irrelevance	91.25	21.67	33.75	34.58
Live				
live_simple	38.37	85.27	86.43	84.11
live_multiple	24.03	79.01	79.87	77.11
live_parallel	12.50	62.50	75.00	75.00
live_par. mult.	0.00	50.00	58.33	58.33
live_irrelevance	82.92	21.04	29.52	33.37
live_relevance	6.25	100.00	100.00	93.75
Multi-turn				
base	0.00	19.00	25.00	14.00
miss_func	0.00	9.50	6.50	1.50
miss_param	0.50	9.50	7.00	1.50
long_context	0.00	10.50	15.50	5.50
Memory				
memory_kv	0.00	0.00	1.29	0.65
memory_rec_sum	0.00	1.94	7.74	1.29
memory_vector	0.00	0.00	5.81	0.65