

Peer Review

Review of: "From Informal to Formal – Incorporating and Evaluating LLMs on Natural Language Requirements to Verifiable Formal Proofs"

Evgenii Arsentev¹

1. Independent researcher

1. What the work claims

The paper starts from the observation that end-to-end benchmarks mix problem-solving, reasoning, and the writing of formal specifications. To separate these skills, the authors split the path from an informal requirement to a verifiable proof into six tasks: requirement analysis (ReqAna), full proof generation (ProofGen), proof segment generation (SegGen), proof completion (ProofComplete), proof infilling (ProofInfill), and proof generation from code (Code2Proof) (Section 2, Figure 2). Five formal languages are covered: Coq, Lean4, Dafny, ACSL, and TLA+ (Section 3.1). Proofs were collected from open-source GitHub repositories and kept only if they verify. They were then split into segments and described in natural language by a GPT-4 family model (Section 3.2; the Abstract names gpt-4o). This gives about 18k instruction-response pairs, divided 8:2 into the fine-tuning set FM-ALPACA (14k) and the benchmark FM-BENCH (about 4k) (Section 3.3, Tables 1 and 2). Formal outputs are scored by running the real verifiers: the Coq, Lean4, and Dafny environments, TLC for TLA+, and Frama-C for ACSL. ReqAna is scored by BLEU (Section 3.4). Ten open models are evaluated without fine-tuning, and three base models of 7B to 8B are fine-tuned (Section 4.1).

The authors report that, without fine-tuning, the models do best on Code2Proof, with an average Pass@1 of 43.57%. They do worst on ProofGen (8.65%) and SegGen (10.61%) (Section 4.2). For SegGen, ACSL is the easiest language (average 34.92%), followed by Dafny (15.97%) (Section 4.3). Fine-tuning on FM-ALPACA lifts the 7B to 8B models to the level of DeepSeek-R1 (27.31% against 27.11%) (Section 4.4). Fine-tuning on

formal data also gives an average increase of 1.37% to 5.15% on math, reasoning, and coding benchmarks (Section 4.5, Table 6).

2. Whether the conclusion holds on the data

What I find valuable is that correctness is decided by the real verifier, not by text similarity. Segments that cannot be checked alone are placed into a proof template with a placeholder and verified as a whole proof (Section 3.4). The UltraChat and Tulu-V3 baselines, added "to distinguish whether the capability improvement is simply because more instruction tuning is applied" (Section 4.1), are a control many dataset papers skip. Table 8 publishes raw failure counts per model, task, and language, which let me check several numbers in the text.

FM-BENCH has only 14 Code2Proof instructions (Table 2, row 6). So the headline that models are best at Code2Proof with an average of 43.57% Pass@1 (Section 4.2) rests on 14 items, and one item is $100/14 = 7.14$ points. The llama3.1-8B exception of 50% on ProofInfill (ACSL) against 21.43% for the 70B model (Section 4.2) is also, by my count, a 14-item cell. Table 8 lists 7 failures for 8B and 11 for 70B, so 7 passes of 14 against 3 of 14, a difference of four items. With four items, I would present the fine-tuning-strategy explanation as a hypothesis.

Coq supplies 14939 of 18293 segments (Table 1), and SegGen supplies 3246 of 3965 instructions (Table 2), about 82% each. It would help to say whether task and model averages (for example, 27.11% for DeepSeek-R1, Section 4.2) are taken over items or over language-task cells. The two can differ a lot when one language holds four fifths of the items. The 12.15% of failures attributed to syntax errors (Section 4.2) needs the same care. In Table 8, for Coq SegGen, the share is 746/2910 for llama3.1-instruct-8B (about 26%) but 9/2894 for deepseek-coder-instruct-7B (about 0.3%). For TLA+ SegGen, it is 66/68 for deepseek-coder-instruct-33B. The keywords in Table 7 also differ in nature: for Lean4, they include "unknown identifier" and "type mismatch," while for Coq, only "Syntax Error:" counts.

The Limitations section discusses leakage only toward the model that generated the descriptions. The proofs come from public GitHub repositories (Appendix D), for Coq a single one, github.com/coq/coq. The split "was applied to proofs" (Section 3.3), not to repositories. So a fine-tuned model sees neighbors of the test proofs, and the base models may have seen the test proofs in pre-training. The gain in Section 4.4 may therefore partly be familiarity with these repositories, and I would add a held-out repository per language to separate the two. Also, ReqAna is scored by BLEU against GPT-4o descriptions (Section 3.4),

and FM-ALPACA responses are GPT-4o descriptions too. The "increase of 41%" (Section 4.4, Table 4) may then measure imitation of GPT-4o wording rather than better analysis.

The Abstract says fine-tuning with formal data "also enhances mathematics, reasoning, and coding capabilities." This rests on one base model, llama3.1-8B, and an average increase of 1.37% to 5.15% (Section 4.5). No repeated runs, seeds, or confidence intervals are reported anywhere in the paper, and gains of this size on one model can sit inside run-to-run variation. Section 4.5 words it carefully ("may indicate"), and I would bring the Abstract to the same level. As rendered on Qeios, Tables 3, 4, 5, 6, and 9 are not legible as text for me, so I could check only numbers quoted in the prose and Tables 1, 2, 7, and 8.

In my reading, the claim that fine-tuning on FM-ALPACA strongly improves in-distribution proof and segment generation is supported. The ranking of tasks, the size-reversal explanation, and the transfer to math, reasoning, and code need more evidence.

3. Reproducibility

The paper gives a good base for reproduction. The fine-tuned models are released at <https://huggingface.co/fm-universe> (Abstract), and the Conclusion says the fine-tuning data and the benchmark are released as well. The Introduction promises a Docker container with validation scripts. Source repositories are listed in Appendix D. Section 4.1 states the fine-tuning hyperparameters (three epochs, learning rate $2e-5$, warm-up ratio 0.04, batch size 512, cosine scheduler) and the decoding settings: greedy with temperature 0.0 for Pass@1, nucleus sampling with temperature 0.2 and five samples, and 3-shot for few-shot. Prompts are shown in Figures 5 to 10 (Appendix F).

For the data, I would add direct links to the dataset, the benchmark, and the Docker image, because the text only points to the Hugging Face organization page. Appendix D gives repository URLs without commit hashes or dates. Since these are live projects, the same extraction run later gives a different set. It would also help to report the versions of Coq, Lean4, Dafny, TLC, and Frama-C, and the verifier time limits, as proof that passes under one version can fail under another. The description generator is called "distilled GPT4" with reference [60] (GPT-4 and GPT-4-turbo Preview) in Section 3.2, but "gpt-4o" in the Abstract and "GPT-4o" in Section 4.1. I would name one model and give its snapshot date.

For models and decoding, no random seeds are reported for fine-tuning, for sampling, or for the random choice of cut lines in ProofComplete and ProofInfill and of one of five instructions (Section 3.2). I would add them. The top-p value for nucleus sampling is missing, and so are the exact checkpoint identifiers of the ten models.

On compute, Section 4.1 says inference runs on a single A100-80G, but DeepSeek-R1 is called "R1-671B" in Section 4.4. It would help to say how this model, and the 70B and 72B ones, were served: locally, quantized, or through an API. Finally, with 14372 FM-ALPACA instructions (Table 2) and a batch size of 512, one epoch is $14372/512 =$ about 28 steps, so three epochs are about 84 optimization steps. I would ask the authors to confirm that 512 is the effective batch size, because large gains from 84 steps are unusual and worth explaining.

4. What specifically to fix

1. Section 4.2 says "we evaluate 8 LLMs," while Section 4.1 and the Abstract say ten, and Table 8 has nine rows, with DeepSeek-R1 absent. I would align the count everywhere and explain why R1 is not in Table 8.
2. Section 4.4 credits 27.31% to "qwen2.5-coder-7B fine-tuned with FM-ALPACA," but Section 4.1 lists the fine-tuned base models as llama3.1-8B, qwen2.5-7B, and deepseek-coder-7B-v1.5. Qwen2.5-coder-7B is not among them, so one of the two places needs a correction.
3. The Abstract says fine-tuning brought "a nearly threefold improvement at most," while Section 4.4 says "SegGen more than triples." It would help to pick one wording and use it in both places.
4. Table 1 gives 118 TLA+ segments in FM-BENCH, yet the SegGen TLA+ cell seems to have 70 items: the 1.43% for llama3.1-8B and 22.86% for llama3.1-70B (Section 4.2) equal $1/70$ and $16/70$, and Table 8 lists 69 and 54 failures there ($69+1 = 70$, $54+16 = 70$). Likewise, Table 1 totals (18293 segments, 14321 and 3972 in the split) differ from Table 2 totals (18337 instructions, 14372 and 3965). I would add a table with the number of test items in every task-language cell and explain how segments map to instructions.
5. For Tables 3, 5, and 9, it would help to add the item count per cell and a confidence interval, at least for cells with 14 items (Code2Proof, Table 2 row 6; ProofInfill ACSL).
6. Please state in Section 4.2 whether averages are over items or over cells, and report both, since Coq holds 14939 of 18293 segments (Table 1).
7. In Section 4.1, Inference Strategies, Pass@1 uses temperature 0.0, and Pass@5 uses 0.2, so the Pass@5 gain in Table 5 mixes two changes: more attempts and a different temperature. The sentence about RQ3 ("a zero-shot greedy search with a temperature of 0.0 and a few-shot nuclear search with a temperature of 0.2 for a fair comparison") does not say which of the two is compared with the fine-tuned models. A Pass@1 at temperature 0.2 would separate the effects. Also, Section

4.3 gives "from 10.82% (Dafny) to 63.64% (ACSL)" without saying whether these are relative gains or percentage points.

8. In the Limitations, second point, the authors admit that in ProofInfill and ProofComplete a model may omit the masked theorem and still pass the verifier, and the remedy is only to put the requirement into the input. I would add a mechanical check that the original theorem or property statement is present in the output, and report how many passes survive it. Commit hashes and verifier versions (Appendix D, Section 3.4) would also help.
9. In Section 4.5 and the Abstract, I would repeat the RQ4 fine-tuning with several seeds and at least one more base model, or soften the Abstract to match the "may indicate" of Section 4.5. For the HumanEval figure of 62.53%, please give the two absolute scores next to it. A held-out repository per language as a leakage control (Appendix D, Section 3.3) would also strengthen this.

5. Recommendation

Minor revisions. The core of the paper is sound, and I would like to say so first: the six-task decomposition (Section 2), the scoring by real verifiers (Section 3.4), the control with general-purpose fine-tuning datasets (Section 4.1), and the released models are all strong points. What remains is reporting and wording, and most of it needs no new experiments. I would reconcile the inconsistent counts and model names in Sections 4.1, 4.2, and 4.4 and in Tables 1, 2, and 8. It would help to give item counts and intervals for small cells, such as the 14 Code2Proof items (Table 2), and to document commits and verifier versions (Appendix D). Last, I would bring the transfer claim in the Abstract in line with Section 4.5, which uses one base model and no repeated runs.

Declarations

Potential competing interests: No potential competing interests to declare.