`miniCTX`: Neural Theorem Proving with (Long-)Contexts

Jiewen Hu, Thomas Zhu, and Sean Welleck

Carnegie Mellon University

Motivation. Formal theorem proving in interactive theorem provers (ITPs) provides a testbed for evaluating the reasoning capabilities of large language models (LLMs). Theorem proving capabilities can then directly translate to automation for mathematicians, such as via tools that complete or formalize proofs [10, 8, 9, 3]. However, despite their promise, we see a gap between current LLM+ITP provers and the complexity of real-world theorem proving.

Our motivating observation is that theorems and proofs depend on various forms of *context*, such as newly-defined definitions and lemmas. For instance, to prove results about a square, one might first formalize a definition of a rectangle, prove some results about rectangles, then specialize them to a newly-defined square.[1] However, existing methods for training and evaluating LLM-based theorem provers do not take such context into account. For example, benchmarks often focus on proving standalone competition problems (e.g., miniF2F [12]) or theorems from a library that the model has trained on (e.g., mathlib [4, 11]), and LLM-based provers are trained to accept only a proof state as input, making them blind to new theorems and definitions [7].

From a language modeling perspective, a key challenge is the sheer size of potentially useful context. For instance, proving a theorem in the Prime Number Theorem project without having seen it during training can require reasoning over a long context of interdependent definitions and theorems [5]. Therefore, theorem proving with newly-defined contexts offers a challenging testbed for long-context reasoning, in addition to more accurately reflecting practical proving.

With these considerations in mind, we develop a new evaluation benchmark and training recipe for theorem proving with newly-defined (long-)contexts. We discuss each in turn below.[2]

Benchmark: `miniCTX`. For evaluation, we develop `miniCTX`, a Lean 4 theorem proving benchmark of theorems that depend on newly-defined lemmas, definitions, and proofs from within a project. `miniCTX` currently consists of 200 theorems sourced from four projects: the PrimeNumberTheorem, Math2001, PFR, and recent results from Mathlib. More broadly, `miniCTX`'s format is amenable to periodically updating the benchmark with new projects to ensure that theorems, definitions, or full projects are not seen by language models trained prior to a particular date. We store the theorem statement, the preceding file contents up to the theorem statement, as well as other metadata (e.g., the Lean version, project commit, position of the theorem), and format the dataset as a JSON-lines file. We make the dataset available on HuggingFace, along with corresponding evaluation code based on the Lean REPL [6].

Training: *file-tuning.* For training, we develop *file-tuning*, a new recipe that fine-tunes a language model to take advantage of new definitions, theorems, and the proof-so-far when generating a proof. Specifically, file-tuning trains with (preceding file context, proof state, next-tactic) tuples instead of the standard approach of training with (proof state, next-tactic) pairs. This lets the model use new definitions, theorems, or other information that are defined prior to the current tactic invocation, and are provided in the model's input context.

---

[1] github.com/AlexKontorovich/PrimeNumberTheoremAnd/PrimeNumberTheoremAnd/Rectangle.lean

[2] Data and models available at `https://huggingface.co/l3lab`. Code will be released prior to the talk.

Method	**Model**	**miniF2F**	**miniCTX-PNT**
State-tactic tuning	`ntp-st-deepseek-coder-1.3b`	32.79% (80/244)	17.02% (8/47)
State-tactic prompting	`llemma-7b`	28.28% (69/244)	29.79% (14/47)
File-tuning	`ntp-context-deepseek-coder-1.3b`	**33.61% (82/244)**	**53.19% (25/47)**

Table 1: Evaluation results on miniF2F and miniCTX's Prime Number Theorem (PNT) split. We evaluate all four models above using the same evaluation code based on the Lean REPL, commit, and lean version across models. As a comparison point, prior work on miniF2F with state-tactic tuning reported 27.9% (LLMstep [10]) and 26.5% (ReProver [11]).

Tools and artifacts. To enable file-tuning and obtaining evaluation data, we implement `ntp-training-data`, a general-purpose data extraction tool that extracts (context, proof state, tactic) examples from an arbitrary Lean 4 repository, and formats them into *instruction tuning* data for LLM fine-tuning. Notably, it runs faster than existing data extraction libraries, taking around 2 hours to extract Mathlib. We make extracted data from Mathlib and its corresponding instruction tuning data publicly available, and plan to extract data from more Lean 4 projects.

We use file-tuning to train a strong baseline for `miniCTX`. Namely, we fine-tune a DeepSeek-Coder-1.3b language model on (context, proof state, tactic) examples extracted with `ntp-training-data`. As an initial method for handling the long context length arising from long Lean files, we either truncate the middle of an input file so that the file contents is 1024 tokens, or take only the preceding 1024 tokens, with the strategy selected at random for each example. Our publicly available data and extraction tool can be used to train larger models, models with longer context windows, or other methods for theorem proving with long contexts.

Performance evaluation. We evaluate models for the task of tactic-based theorem proving: given a theorem statement, a model generates one tactic at a time within a best-first search. We use standard settings for the best-first search. We evaluate three types of baselines on `miniCTX`:

1. *State-tactic tuning*: the standard approach to fine-tuning a language model for theorem proving. We finetune a DeepSeek-1.3B[2] model on state-tactic pairs extracted with `ntp-training-data`. It outperforms previous openly available (state, tactic) models.

2. *State-tactic* prompting: we prompt a pretrained LLM with (state, tactic) examples, following [1]. We use the Llemma-7b model [1].

3. *File-tuning*: we provide our file-tuned DeepSeek-1.3B model with a (file context, state) pair as input, and append each generated tactic to the file context during best-first search.

First, we evaluate on miniF2F, the de-facto standard benchmark for neural theorem proving. We adapt it to the context-dependent setting with a context consisting of the miniF2F import statements. The state-tactic tuned models achieve 32.79%, which is higher than existing methods tuned on LeanDojo [11]. This validates the efficacy of `ntp-training-data`'s extraction. The file-tuned model achieves 33.61%, showing that file-tuning maintains or even improves performance in a setting that does not require context (miniF2F competition problems).

Second, we show results on the PrimeNumberTheorem subset of `miniCTX`. These proofs typically depend on more context than those in miniF2F. We see a dramatic improvement for the file-tuned model over the state-tactic methods. The gap shows the importance of evaluating the context-dependent setting, and the importance of neural theorem proving methods that leverage context beyond the proof state.

References

[1] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. M. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024.

[2] D. Y. Z. X. K. D. W. Z. G. C. X. B. Y. W. Y. L. F. L. Y. X. W. L. Daya Guo, Qihao Zhu. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024.

[3] S. Gadgil. Leanaide. `https://github.com/siddhartha-gadgil/LeanAide`, 2024.

[4] J. M. Han, J. Rute, Y. Wu, E. Ayers, and S. Polu. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

[5] A. Kontorovich. Prime number theorem and. `https://github.com/AlexKontorovich/PrimeNumberTheoremAnd`, 2024.

[6] Lean Prover Community. repl. `https://github.com/leanprover-community/repl`, 2024.

[7] S. Polu and I. Sutskever. Generative language modeling for automated theorem proving, 2020.

[8] P. Song, K. Yang, and A. Anandkumar. Towards large language models as copilots for theorem proving in Lean. *arXiv preprint arXiv: Arxiv-2404.12534*, 2024.

[9] S. Welleck. Llmlean. `https://github.com/cmu-l3/llmlean`, 2024.

[10] S. Welleck and R. Saha. Llmstep: Llm proofstep suggestions in lean. *arXiv preprint arXiv:2310.18457*, 2023.

[11] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*, 2023.

[12] K. Zheng, J. M. Han, and S. Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

Appendix

Example 1: MiniF2F

```
theorem amc12b_2020_p2 :
    (100 ^ 2 - 7 ^ 2 : ℝ) / (70 ^ 2 - 11 ^ 2) * ((70 - 11) * (70 + 11)
    / ((100 - 7) * (100 + 7))) =
      1 := by
  ring
```

Example 2: PrimeNumberTheoremAnd

```
lemma RectSubRect {x0 x1 x2 x3 y0 y1 y2 y3 : ℝ} (x0_le_x1 : x0 ≤ x1)
    (x1_le_x2 : x1 ≤ x2)
    (x2_le_x3 : x2 ≤ x3) (y0_le_y1 : y0 ≤ y1) (y1_le_y2 : y1 ≤ y2)
    (y2_le_y3 : y2 ≤ y3) :
    Rectangle (x1 + y1 * I) (x2 + y2 * I) ⊆ Rectangle (x0 + y0 * I) (x3 +
     y3 * I) := by sorry

lemma RectSubRect' {z0 z1 z2 z3 : ℂ} (x0_le_x1 : z0.re ≤ z1.re)
    (x1_le_x2 : z1.re ≤ z2.re)
    (x2_le_x3 : z2.re ≤ z3.re) (y0_le_y1 : z0.im ≤ z1.im) (y1_le_y2 :
    z1.im ≤ z2.im)
    (y2_le_y3 : z2.im ≤ z3.im) :
    Rectangle z1 z2 ⊆ Rectangle z0 z3 := by
  rw [← re_add_im z0, ← re_add_im z1, ← re_add_im z2, ← re_add_im z3]
  exact RectSubRect x0_le_x1 x1_le_x2 x2_le_x3 y0_le_y1 y1_le_y2 y2_le_y3
```

Figure 1: Two contrasting examples of Lean 4 code from different repositories. Example 1 is from the miniF2F repository, which contains problems from high school math competitions that are typically independent and self-explanatory. Example 2 is from the PrimeNumberTheoremAnd project, which often has many dependencies. For instance, the proof of `RectSubRect'` depends on `RectSubRect`.

Input Prompt

```
/- You are proving a theorem in Lean 4.
You are given the following information:
- The file contents up to the current tactic, inside [CTX]...[/CTX]
- The current proof state, inside [STATE]...[/STATE]

Your task is to generate the next tactic in the proof.
Put the next tactic inside [TAC]...[/TAC]
-/
[CTX]
... (truncated)

/-- A ‘Rectangle‘ has corners ‘z‘ and ‘w‘. -/
def Rectangle (z w : ℂ) : Set ℂ := [[z.re, w.re]] ×ℂ [[z.im, w.im]]

... (truncated)

lemma RectSubRect {x₀ x₁ x₂ x₃ y₀ y₁ y₂ y₃ : ℝ} (x₀_le_x₁ : x₀ ≤ x₁)
    (x₁_le_x₂ : x₁ ≤ x₂)
    (x₂_le_x₃ : x₂ ≤ x₃) (y₀_le_y₁ : y₀ ≤ y₁) (y₁_le_y₂ : y₁ ≤ y₂)
    (y₂_le_y₃ : y₂ ≤ y₃) :
    Rectangle (x₁ + y₁ * I) (x₂ + y₂ * I) ⊆ Rectangle (x₀ + y₀ * I) (x₃ +
     y₃ * I) := by
  rw [rect_subset_iff, mem_Rect, mem_Rect]
  refine ⟨⟨?_, ?_, ?_, ?_⟩, ?_, ?_, ?_, ?_⟩
  all_goals simpa using by linarith

lemma RectSubRect’ {z₀ z₁ z₂ z₃ : ℂ} (x₀_le_x₁ : z₀.re ≤ z₁.re)
    (x₁_le_x₂ : z₁.re ≤ z₂.re)
    (x₂_le_x₃ : z₂.re ≤ z₃.re) (y₀_le_y₁ : z₀.im ≤ z₁.im) (y₁_le_y₂ :
    z₁.im ≤ z₂.im)
    (y₂_le_y₃ : z₂.im ≤ z₃.im) :
    Rectangle z₁ z₂ ⊆ Rectangle z₀ z₃ := by
[/CTX]
[STATE]
zw: ℂ
c: ℝ
z₀z₁z₂z₃: ℂ
x₀_le_x₁: z₀.re ≤ z₁.re
x₁_le_x₂: z₁.re ≤ z₂.re
x₂_le_x₃: z₂.re ≤ z₃.re
y₀_le_y₁: z₀.im ≤ z₁.im
y₁_le_y₂: z₁.im ≤ z₂.im
y₂_le_y₃: z₂.im ≤ z₃.im
⊢ Rectangle z₁ z₂ ⊆ Rectangle z₀ z₃
[/STATE]
[TAC]
```

Proof Generated by ntp-context-deepseek-coder-1.3b

```
simpa using RectSubRect x₀_le_x₁ x₁_le_x₂ x₂_le_x₃ y₀_le_y₁ y₁_le_y₂
    y₂_le_y₃
[/TAC]
```

Figure 2: Illustration of a tactic prediction task using **File-tuning**. The top box shows the input prompt, which includes the context and the current proof state. The bottom box displays the proof tactic generated by the `ntp-context-deepseek-coder-1.3b` model.