Carnegie Mellon University

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

Jiewen Hu, Thomas Zhu, Sean Welleck

# Background

**Current Lean Datasets:**

- Competition problems: minif2f, ProofNet primarily offer standalone competition problems
- Mathlib4: LeanStep, LeanDojo
- Other context datasets: datasets like CoqGym attempt to gather all possible data from the internet, but this exhaustive approach risks data contamination.

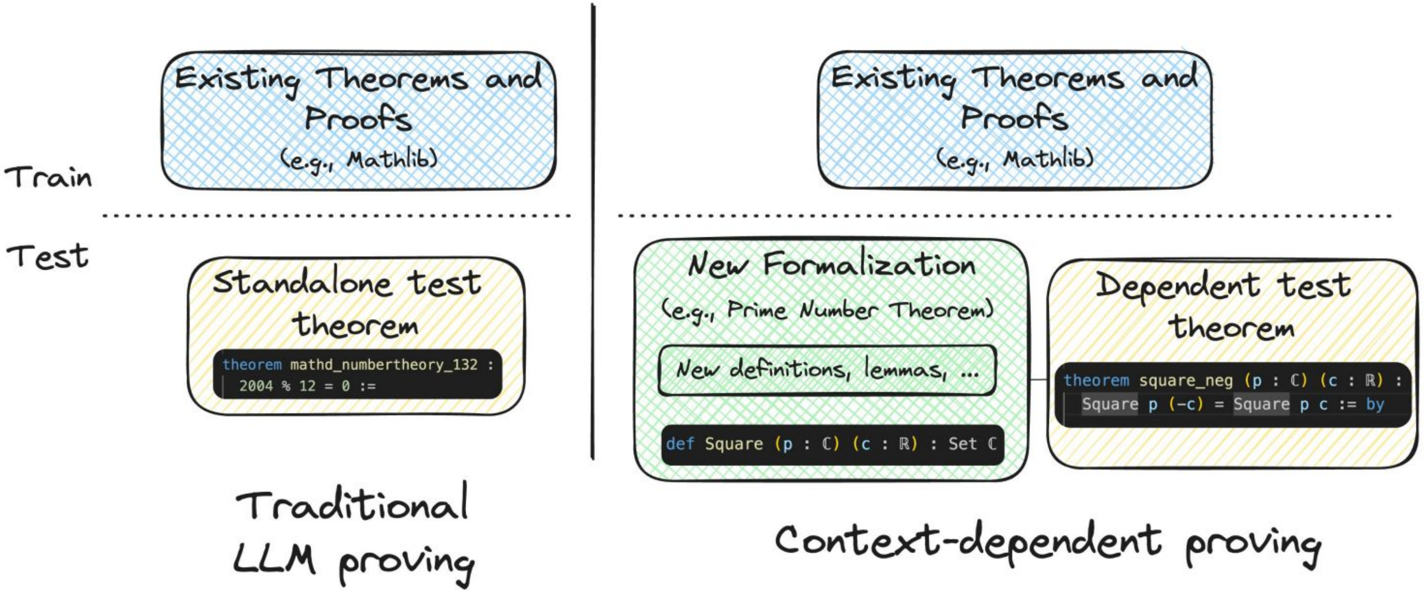
**Tactic Suggestion Tools:**

- Tools like LeanCopilot and llmLean are being developed to act as copilots for formal proofs.
- Take file contexts as inputs, which can include previous definitions, lemmas, and human-written comment

Carnegie Mellon University

# Background

**Context-dependent proving:** Current theorem-proving datasets (e.g., minif2f, proofnet) focus on standalone problems




Carnegie Mellon University

# miniCTX benchmark

Each theorem in miniCTX is accompanied by the following data, formatted in JSON:

1. **Theorem statement,**
2. **Preceding file contents up to the theorem statement,**
3. Metadata, including:

   (a) File name,

   (b) **Project commit and version**,

   (c) **Commit at which the theorem and its file was added**,

   (d) Position of the theorem and number of premises preceding it,

   (e) Proof length and type,

   (f) **Whether the statement or proof uses new definitions or lemmas from the file or repository.**

Carnegie Mellon University

# miniCTX benchmark

```
import Mathlib.Data.Real.Basic

/-!
# Square function
We define the squaring function `s : ℝ → ℝ` to be `s x := x * x`.
-/

def s (x : ℝ) : ℝ := x * x

lemma s_eq_pow_two {x : ℝ} : s x = x ^ 2 := by
  rw [s, pow_two]
```

Original Lean File


```
{
  "srcContext": "import Mathlib.Data.Real.Basic\n\n/-!\n# Square function\nWe define the squaring fu
  "theoremStatement": "lemma s_eq_pow_two {x : ℝ} : s x = x ^ 2",
  "theoremName": "s_eq_pow_two",
  "fileCreated": "(git commit)",
  "theoremCreated": "(git commit)",
  "file": "(file name)",
  "positionMetadata": {
    "lineInFile": 10,
    "tokenPositionInFile": 152,
    "theoremPositionInFile": 1
  },
  "dependencyMetadata": {
    "inFilePremises": true,
    "repositoryPremises": false
  },
  "proofMetadata": {
    "hasProof": true,
    "proof": "by\n  rw [s, pow_two]",
    "proofType": "tactic",
    "proofLengthLines": 2,
    "proofLengthTokens": 20
  }
}
```


miniCTX problem

Carnegie Mellon University

# miniCTX benchmark

Sources:

- **Prime Number Theorem Split**
  - Contains theorems related to “rectangle”, capturing newly defined concepts and related operations
- **PFR (Polynomial Freiman–Ruzsa) Split:**
  - Captures challenging, long proofs with extensive in-file and cross-file dependencies
- **Recent Mathlib Split:**
  - Represents a popular library environment
- **HTPI (How to Prove It) Split:**
  - Derived from a textbook environment, combining definitions, sample lemmas, and exercises

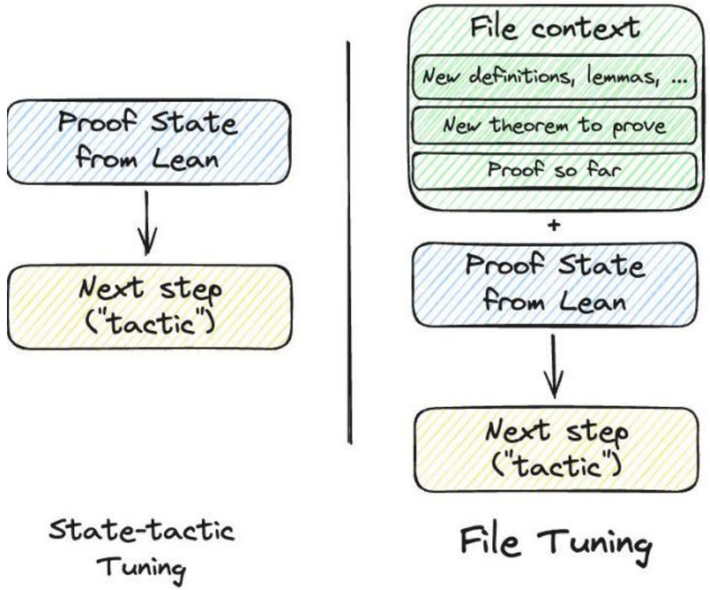
| | Split | Problems | Avg. Context Length (tokens) | Avg. Proof Steps |
|---|---|---|---|---|
| miniF2F [[6]] | Valid/Test | 488 | 153* | 3.0** |
| miniCTX | Prime | 87 | 10,630 | 3.6 |
| | PFR | 54 | 17,495 | 27.7 |
| | Mathlib | 50 | 14,440 | 6.1 |
| | HTPI | 185 | 39,050 | 10.7** |
| | **All** | 376 | 26,106 | 10.9 |

# Why miniCTX?

1. **Context-dependent proving**
2. **Generalization**: evaluate models generalization ability from different levels:
   - Theorem-Level: the proof must not occur in the model’s training data
   - Context-Level: the context and proof must not occur in the training data
   - Project-Level: the entire repository must not occur in the training data.
3. **Active Updates:** miniCTX is designed as a dynamic benchmark that can be updated regularly using our automated annotation and extraction toolkit.

# Baselines

1. File-Tuning (trained on mathlib):
   - Input: context information + current state
   - Output: next tactic
2. State-Tactic Tuning:
   - Input: current state
   - Output: next tactic
3. GPT-4o (with and without context)
4. Llemma-7b



# Performance Comparison

File aware methods consistently outperformed other baseline methods across all splits

| | MiniF2F | MiniCTX | | | | |
|---|---|---|---|---|---|---|
| **Models** | **Test** | **Prime** | **PFR** | **Mathlib** | **HTPI** | **Avg.** |
| GPT-4o (full proof) | - | 1.15% | 5.56% | 2.00% | 9.73% | 5.59% |
| GPT-4o (+ context) | - | 13.79% | 1.85% | 18.00% | 31.89% | 22.07% |
| State-tactic prompting | 28.28% | 19.54% | 5.56% | 16.00% | 19.15% | 20.61% |
| State-tactic tuning | 32.79% | 11.49% | 5.56% | 22.00% | 5.95% | 9.31% |
| File tuning | **33.61%** | **32.18%** | **5.56%** | **34.00%** | **38.38%** | **31.65%** |

Table 2: Performance comparison of different models on MiniF2F and MiniCTX.

Carnegie Mellon University

# Performance Comparison

File-tuning especially helps on problems with dependencies

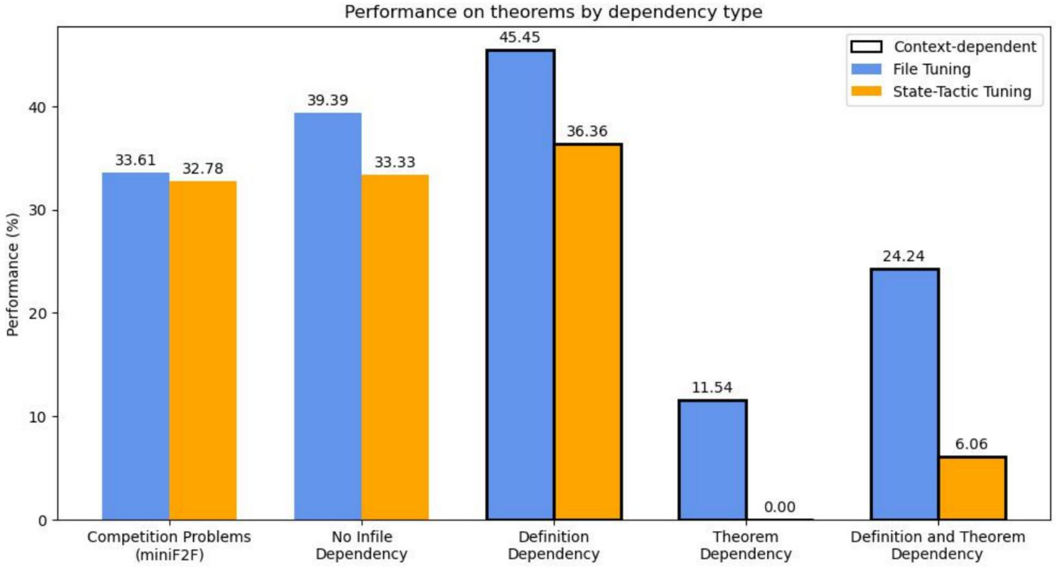



Figure 3: Performance partitioned by dependency type. File-tuned models substantially outperform state-tactic tuned models on theorems with definition and/or theorem dependencies.

# Interesting Findings

1. Some definition dependencies can be handled given access only to the proof state

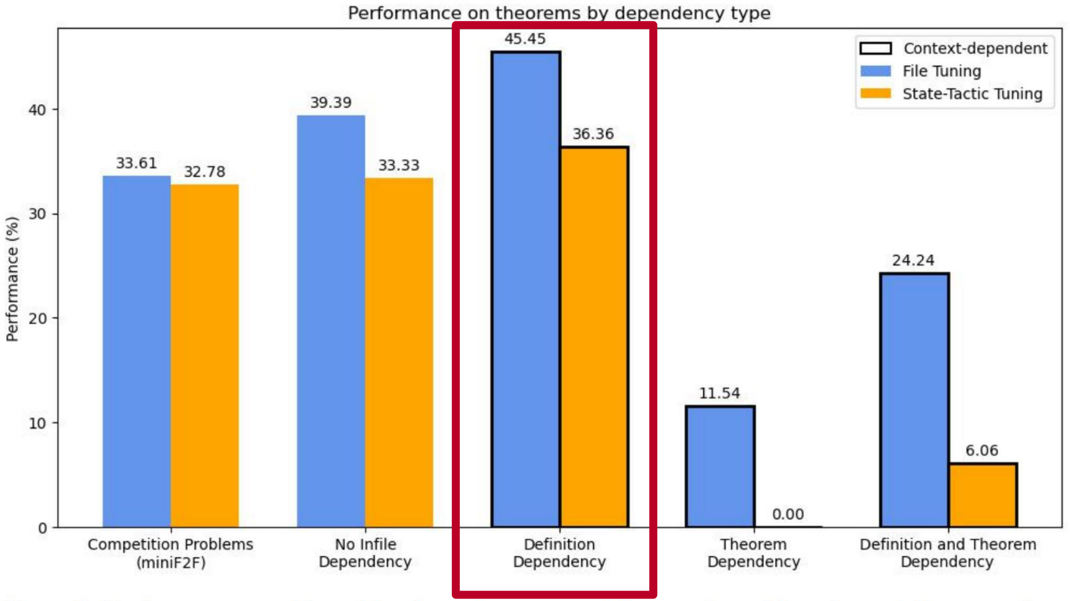



Figure 3: Performance partitioned by dependency type. File-tuned models substantially outperform state-tactic tuned models on theorems with definition and/or theorem dependencies.

Carnegie Mellon University

# Interesting Findings

1. Some definition dependencies can be handled given access only to the proof state
2. Definitions and theorems in the context are both important.
3. Pass rate was similar without proofs in the context (but different sets of solved problems)

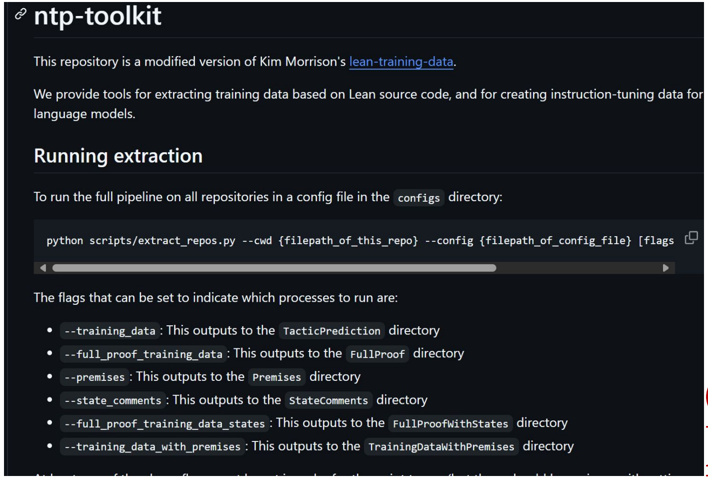
| Context Type | Accuracy (%) |
|---|---|
| No Context | 17.02 |
| Imports & Definition | 29.79 |
| Theorems | 38.29 |
| No proof | 46.81 |
| All | 46.81 |

Table 3: Ablating components of the context.

# Toolkit and Resources

**NTP-Toolkit:** based on Kim Morrison's work, is designed for extracting and annotating data from Lean source code

- Easily extract the data with annotations by running: python scripts/extract_repos.py --cwd {path_to_repo} --config {path_to_config_file} --training_data
- Convert into instruction-tuned data



## ntp-toolkit

This repository is a modified version of Kim Morrison's lean-training-data.

We provide tools for extracting training data based on Lean source code, and for creating instruction-tuning data for language models.

### Running extraction

To run the full pipeline on all repositories in a config file in the `configs` directory:

```
python scripts/extract_repos.py --cwd {filepath_of_this_repo} --config {filepath_of_config_file} [flags
```

The flags that can be set to indicate which processes to run are:

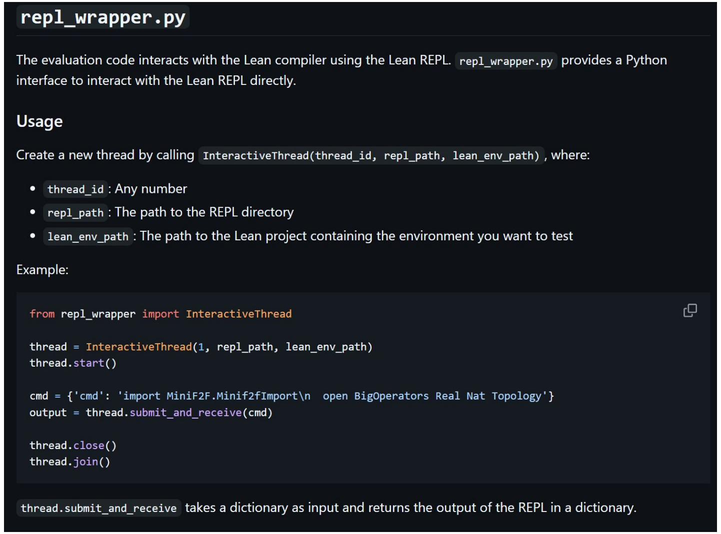
- `--training_data` : This outputs to the `TacticPrediction` directory
- `--full_proof_training_data` : This outputs to the `FullProof` directory
- `--premises` : This outputs to the `Premises` directory
- `--state_comments` : This outputs to the `StateComments` directory
- `--full_proof_training_data_states` : This outputs to the `FullProofWithStates` directory
- `--training_data_with_premises` : This outputs to the `TrainingDataWithPremises` directory

Carnegie Mellon University

# Toolkit and Resources

**REPL-Wrapper**: a Python wrapper for the Lean REPL

**ntp-mathlib-instruct-context Dataset:** extracted and converted instruction-tuned data from mathlib using our toolkit



## repl_wrapper.py

The evaluation code interacts with the Lean compiler using the Lean REPL. `repl_wrapper.py` provides a Python interface to interact with the Lean REPL directly.

### Usage

Create a new thread by calling `InteractiveThread(thread_id, repl_path, lean_env_path)`, where:

- `thread_id` : Any number
- `repl_path` : The path to the REPL directory
- `lean_env_path` : The path to the Lean project containing the environment you want to test

Example:

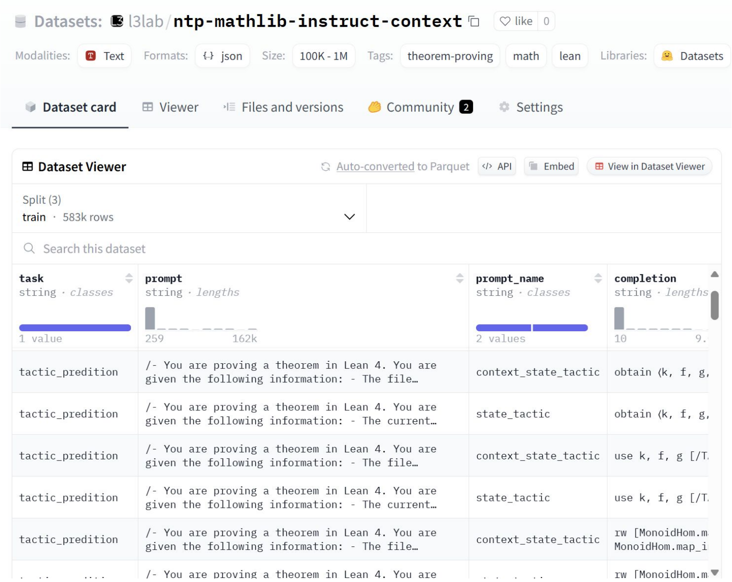
```
from repl_wrapper import InteractiveThread

thread = InteractiveThread(1, repl_path, lean_env_path)
thread.start()

cmd = {'cmd': 'import MiniF2F.Minif2fImport\n  open BigOperators Real Nat Topology'}
output = thread.submit_and_receive(cmd)

thread.close()
thread.join()
```

`thread.submit_and_receive` takes a dictionary as input and returns the output of the REPL in a dictionary.



Datasets: l3lab/**ntp-mathlib-instruct-context** like 0

Modalities: Text Formats: json Size: 100K - 1M Tags: theorem-proving math lean Libraries: Datasets

Dataset card | Viewer | Files and versions | Community 2 | Settings

**Dataset Viewer** Auto-converted to Parquet API Embed View in Dataset Viewer

Split (3)
train · 583k rows

Search this dataset

| task<br>string · classes<br>1 value | prompt<br>string · lengths<br>259 162k | prompt_name<br>string · classes<br>2 values | completion<br>string · lengths<br>10 9. |
|---|---|---|---|
| tactic_predition | /- You are proving a theorem in Lean 4. You are given the following information: - The file… | context_state_tactic | obtain ⟨k, f, g, |
| tactic_predition | /- You are proving a theorem in Lean 4. You are given the following information: - The current… | state_tactic | obtain ⟨k, f, g, |
| tactic_predition | /- You are proving a theorem in Lean 4. You are given the following information: - The file… | context_state_tactic | use k, f, g [/T. |
| tactic_predition | /- You are proving a theorem in Lean 4. You are given the following information: - The current… | state_tactic | use k, f, g [/T. |
| tactic_predition | /- You are proving a theorem in Lean 4. You are given the following information: - The file… | context_state_tactic | rw [MonoidHom.m MonoidHom.map_i |
| | /- You are proving a theorem in Lean 4. You are | | rw [MonoidHom.m |

Carnegie Mellon University

# Contributions

1. **miniCTX benchmark:** first benchmark aimed at the real-world theorem proving
2. **Ntp-toolkit:** automate the extraction and annotation of theorem-proving data
3. **Lean REPL Wrapper:** Lean REPL wrapper to simplify interactions with Lean
4. **File-Tuning:** a strong baseline method for training models using full file contexts
5. **ntp-mathlib-instruct-context Dataset:** training data that includes in-file context information

# Next Step

1. Extend the benchmark to areas beyond math:
   - program verification: Formal Proof and Verification by Brown University
   - scientific computing: SciLean
2. Evaluate premise selection
3. Crossfile: add new splits for crossfile dependency