Lean at scale

Caching, reproducible projects,
and interactive tooling

What happens when you import Mathlib?

Does Lean rerun every tactic
used to prove the imported library?

```
import Mathlib.Algebra.Group.Basic
```

Compiled modules and caches

.lean

Source declarations and proof scripts

.olean

Compiled module information used by importing files

Cache download

Reuse build results produced on another machine
(if not from a trusted source - be careful!!!)

A reproducible Lean project

File	Purpose
lean-toolchain	Lean version
lakefile.toml	Dependencies, targets, and build settings
lake-manifest.json	Exact resolved dependency commits
.lake/	Downloaded sources and generated build outputs

Typically you shouldn’t commit the .lake folder

The lakefile and the manifest

You write lakefile.toml, and then Lake generates lake-manifest.json.

When…	Lake uses…
A manifest exists	TOML settings and the recorded dependency revisions
The manifest is missing	Dependency requests to generate a resolution
You run lake update	Dependency requests to refresh the manifest

Three different jobs

Command	Job
lake update	Resolve dependencies and refresh the manifest
lake exe cache get	Restore Mathlib’s supported compiled outputs
lake build	Bring the requested build targets up to date

Builds can reuse local outputs and, when configured, remote artifacts (e.g. for Mathlib Cache, or from Reservoir).

cache demo

You can find cached builds on Reservoir

Look for all three:

The exact package revision
The matching Lean toolchain
A downloadable build archive with a size

Example: NavierStokesAndEuler 0.1.0 ~ 815 MB (13M loc)

Reservoir: NavierStokesAndEuler versions

Reservoir archives

```
[[require]]
name = "NavierStokesAndEuler"
scope = "openai"
version = "0.1.0"
```

```
# Resolve the new dependency and update lake-manifest.json
lake update NavierStokesAndEuler

# Download Mathlib's compiled artifacts, since this package uses Mathlib
lake exe cache get

# Explicitly fetch the package's Reservoir build archive
lake build @NavierStokesAndEuler:barrel

# Build your project, reusing compatible downloaded artifacts
lake build
```

The Language Server Protocol

LSP = Language Server Protocol

Editor action	Language service
Hover on a name	Type information and documentation
Change the document	Diagnostics for the changed text
Move through a Lean proof	Goal information through Lean extensions

Feedback before saving

```
theorem editor_demo (P Q : Prop) (h : P ∧ Q) : Q ∧ P := by
  constructor
  · exact h.2
  · exact h.1
```

Change the final h.1 to h.2 without saving.

How can Lean report an error before a terminal build runs?

The Lean REPL

REPL = Read–Eval–Print Loop

A program sends a Lean command or tactic
and receives results it can inspect.

```
example (P Q : Prop) (h : P ∧ Q) : Q ∧ P := by
  sorry
```

The REPL can return an ID for the proof state at the placeholder.

Two attempts from the same state

h : P ∧ Q
⊢ Q ∧ P

{"proofState": 0, "tactic": "exact True.intro"}

Type mismatch: True is not Q ∧ P

{"proofState": 0, "tactic": "exact ⟨h.2, h.1⟩"}

No remaining goals

The Model Context Protocol

MCP = Model Context Protocol

An AI application discovers available tools
and calls them through a common interface.

Each tool advertises a name, a description,
and the arguments it accepts.

Lean tools available through MCP

Tool	Result
lean_goal	The goal at a source position
lean_diagnostic_messages	Errors and warnings in a Lean file
lean_multi_attempt	Results of several tactic attempts

A Lean MCP server can use LSP or a REPL underneath.

lean-lsp-mcp project and documentation

A tool-assisted proof attempt

h : P ∧ Q
⊢ P

Attempt	Lean’s feedback
exact h.2	Fails: h.2 has type Q
exact h.1	Succeeds: h.1 has type P