

What different between leandojo and jixia

Short answer:

- **LeanDojo** → mature *AI training + evaluation framework + dataset pipeline* for Lean theorem proving.
 - **Jixia** → newer *Lean static analysis / data extraction + IDE support tool* (more infrastructure than full training framework).
-

LeanDojo vs Jixia — Core Difference

1 LeanDojo

Position: End-to-end AI theorem-proving training ecosystem

Main goal:

- 👉 Turn Lean repositories into ML datasets
- 👉 Train / evaluate / deploy theorem-proving AI agents

What it provides

- Extracts **proof states, tactics, premises, ASTs** from Lean repos
- Provides **benchmarks + datasets** (100k-scale theorem proofs)
- Provides **gym-like environment** so AI can interact with Lean proof state
- Supports **training loops** (SFT, RL, retrieval-augmented proving)

Evidence:

- LeanDojo is an end-to-end framework for training and deploying AI theorem provers.
leandojo.org +1
- It extracts structured proof data (AST, tactics, premises, dependencies).
leandojo.org
- Dataset scale example: ~98k theorems + proof annotations from mathlib.
Abaka...
- It turns Lean into a gym-style environment for RL / agent interaction.
leandojo.org

👉 In practice:

If you want to **train a Lean LLM prover**, LeanDojo is the default choice.

2 Jixia

Position: Lean static analysis + data extraction + IDE-enabling infrastructure

Main goal:

- 👉 Provide Lean-aware analysis
- 👉 Extract data for ML pipelines
- 👉 Support Lean tooling (IDE, analysis)

What it provides

- Static analysis of Lean 4 source
- Non-intrusive analysis (no modification of source files)
- Data extraction for downstream ML systems
- Part of PKU AI-for-math ecosystem

Evidence:

- Jixia is a static analysis tool for Lean 4 aimed at building Lean-aware IDEs and extracting ML data. [GitHub](#)
- Designed to work without modifying target Lean code (good for large mathlib caching). [GitHub](#)

Also appears in research infra:

- Jixia-interactive environment used for data synthesis / training pipelines in some Lean prover work. [arXiv](#)

👉 In practice:

If you want to **build tools / pipelines / custom datasets**, Jixia is more "infrastructure layer".

❖ Mental Model (Stack Layer)

AI Training Stack (Simplified)

Layer	Tool
Prover models	ReProver / REAL-Prover / custom LLM
Training framework	⭐ LeanDojo
Data extraction infra	LeanDojo OR Jixia
Static analysis / IDE infra	⭐ Jixia
Raw Lean code	mathlib / GitHub repos

⚖️ Practical Comparison

Dimension	LeanDojo	Jixia
Maturity	⭐ High (papers + benchmarks + ecosystem)	Medium / emerging
Scope	Full ML training loop	Static analysis + infra
Dataset	Built-in benchmarks	Helps generate custom data
RL / agent training	Yes	No (indirect)
IDE tooling	Not focus	Yes (goal)
Best use	Train theorem proving AI	Build Lean data / tools / pipeline infra

🧭 Which Should You Use?

Based on typical AI-math workflows:

If you want:

- Train Lean theorem-proving LLM
- Reproduce NeurIPS / SOTA pipelines

👉 Use LeanDojo

If you want:

- ✓ Build new dataset extraction pipeline
 - ✓ Build Lean IDE / tooling / static analysis
 - ✓ Custom data generation infra
- 👉 Consider Jixia
-

💡 Real-World Analogy

- LeanDojo = PyTorch Lightning for Lean AI
 - Jixia = Clang static analyzer for Lean
-

If you want, I can go deeper, for example:

- Dataset quality comparison (very interesting)
- Which one big labs actually use
- Which is better for your HunyuanProver-style data generation (I noticed you work near this space)

Just tell me 👍

🔗 🤗 ⚡ ⌛ ... 🔍 Sources

translate to Chinese

🔗 🖊

简短结论：

- LeanDojo → 成熟的 AI 训练 + 评测框架 + 数据集管道 (偏完整训练生态)
 - Jixia → 较新的 Lean 静态分析 / 数据抽取 + IDE 支持工具 (偏基础设施层)
-

LeanDojo vs Jixia —— 核心区别

1 LeanDojo

定位： 端到端 AI 定理证明训练生态

主要目标：

- 👉 把 Lean 仓库转成机器学习数据集
- 👉 训练 / 评测 / 部署定理证明 AI

提供能力

- 提取 证明状态、tactic、前提、AST
- 提供 基准数据集 + Benchmark (10万级 theorem)
- 提供 类似 Gym 的交互环境 (AI 可与 Lean 证明状态交互)
- 支持 训练流程 (监督学习、强化学习、检索增强证明)

👉 实际用途：

如果你要 训练 Lean LLM Prover，LeanDojo 基本是默认选择。

2 Jixia

定位： Lean 静态分析 + 数据抽取 + IDE 基础设施

主要目标：

- 👉 Lean 代码分析
- 👉 为 ML 提供数据抽取能力
- 👉 支持 Lean IDE / 工具链

提供能力

- Lean4 静态分析
- 不需要修改原 Lean 代码
- 支持 ML 数据抽取
- 北大 AI for Math 体系中的基础设施组件

👉 实际用途：

如果你要 搭数据管道 / 做工具链 / 自定义数据生成，Jixia 更偏底层基础设施。

技术栈层级理解

AI 数学证明技术栈（简化）

层级	工具
Prover 模型	ReProver / REAL-Prover / 自研 LLM
训练框架	 LeanDojo
数据抽取基础设施	LeanDojo 或 Jixia
静态分析 / IDE 基础设施	 Jixia
原始代码	mathlib / GitHub Lean 项目

实际对比

维度	LeanDojo	Jixia
成熟度	 高（论文 + Benchmark + 生态）	中等 / 新兴
覆盖范围	完整训练闭环	静态分析 + 基础设施
数据集	内置 Benchmark	更偏自定义数据生成
RL / Agent 训练	支持	不直接支持
IDE 工具	不是重点	是目标方向
最适合	训练 AI 定理证明	构建 Lean 数据 / 工具基础设施

怎么选？

如果你想：

👉 选 LeanDojo

如果你想：

- ✓ 构建 Lean 数据生成管道
- ✓ 做 Lean IDE / 静态分析工具
- ✓ 自定义数据抽取基础设施

👉 看 Jixia

💡 类比理解

- LeanDojo ≈ Lean 领域的 PyTorch Lightning
- Jixia ≈ Lean 领域的 Clang 静态分析器

如果你愿意，我可以再深入，比如：

- 两者 数据质量差异
- 大模型团队真实使用情况
- 哪个更适合做 数学推理训练数据生成（比如 prover 数据工厂）

你想往哪个方向深入？



两者 数据质量差异

LeanDojo vs Jixia —— 数据质量差异（核心结论）

我直接给你结论级别对比（偏 AI 训练数据质量角度）：

维度	LeanDojo	Jixia
数据结构完整性	★★★★★ 非常完整	★★★ 中等
语义上下文	★★★★★ 有完整证明轨迹	★★ 多为结构信息
噪声控制	★★★★★ 较低	★★ 依赖解析质量
AI 训练适配度	★★★★★ 专门设计	★★★ 更偏工具数据
可复现性	★★★★★ 高	★★★ 中

一、LeanDojo：偏高质量 AI 训练级数据

✓ 优势本质

LeanDojo 是专门为 AI 训练 theorem proving 设计的，不只是导出 Lean 代码，而是构造：

- tactic 执行轨迹
- proof state 演化
- 依赖关系图
- 前提检索训练样本

例如 LeanDojo benchmark 已经被用于论文评测 proving 模型性能，并成为标准评测环境之一。arXiv

👉 说明：

- 数据是 语义 + 结构 + 动态执行信息
- 对 LLM / RL / agent training 非常友好

✓ 数据质量特点

LeanDojo 数据通常包含：

CSS

 Copy code

State_before

Tactic

`State_after`
`Dependency graph`
`Proof tree path`

这意味着：

- AI 能学习 **推理过程**
 - 不只是学习“定理文本 → 定理文本”
-

二、Jixia：偏结构化导出 / 分析级数据

✓ 优势本质

Jixia 更偏：

- Lean 代码结构解析
- AST / 定理依赖
- 静态分析结果
- 项目级索引

👉 更像：

sql

 Copy code

Lean repo → structured dataset → `search` / `indexing` / `tooling`

✓ 数据质量特点

通常偏：

- theorem metadata
- dependency graph
- AST structure
- symbol usage

但通常缺：

- tactic step trace
- proof state evolution
- execution runtime context

三、最关键质量差异（真正 AI 训练差别）

🧠 如果训练 Proof LLM

LeanDojo 优势巨大，因为：

能力	LeanDojo	Jixia
学 tactic 选择	✓	✗
学 proof planning	✓	△
学 step reasoning	✓	✗
只学 theorem embedding	✓	✓

🧠 如果做知识库 / 搜索 / 检索

Jixia 其实足够甚至更轻量：

- repo index
- semantic search
- dependency query

四、用一句话总结

👉 LeanDojo = 训练 Proof AI 的高质量行为数据

👉 Jixia = Lean 数学库结构化知识数据

五、如果用“数据金字塔”比喻

markdown

 Copy code

LeanDojo
(行为 + 语义 + 执行)

↑
Jixia
(结构 + 语法 + 依赖)

六、如果你是做 AI 产品（我猜你偏这个方向）

经验建议：

如果目标是：

- ✓ 自动写 proof
- ✓ RL proof agent
- ✓ reasoning LLM

👉 LeanDojo 更核心

如果目标是：

- ✓ 数学搜索
- ✓ theorem embedding
- ✓ Lean IDE 智能补全

👉 Jixia 就够

如果你愿意，我可以给你一个**更狠一点的业内真实评价版本**（比如：

- 哪个更可能成为未来标准数据层
- 哪个更适合商业化
- 哪个更容易 scale 到 mathlib4 全量）

要不要我直接给你这个层级分析？

How to make one product with both benefits

这个问题其实非常好，而且已经是 **下一代 AI Math 基础设施产品** 在做的方向。
我直接给你一个 **可落地产品架构思路**，不是理论。

🧠 目标：同时拿到两者优势

你想要融合：

- LeanDojo → 高质量行为级训练数据 (proof trajectory)
 - Jixia → 大规模结构化知识索引 (静态分析 + repo级理解)
-

★ 最优产品思路（一句话）

- 👉 用 Jixia 做底层知识图谱和全库索引
 - 👉 用 LeanDojo 做行为级 proof 训练数据生成层
-

🏗 推荐产品架构 (Industry-grade)

Layer 1 — Raw Lean Infra Layer

(类似 Data Lake)

来自：

- mathlib4
- GitHub Lean repos
- internal math repos

工具：

- Jixia parser
- Lean server AST export
- dependency graph builder

产出：

mathematica

 Copy code

Symbol Graph

Dependency Graph

AST Graph

Module Index

Theorem Metadata

👉 这一层解决：

- 可扩展
 - 全库扫描
 - 快速更新
 - IDE / search / embedding
-

Layer 2 — Semantic Knowledge Graph Layer

(Jixia 强项放大)

构建：

nginx

 Copy code

Theorem → Depends on → Theorem

Theorem → Uses → Symbol

Theorem → Category → Algebra / Topology / etc

增强：

- embedding
 - retrieval index
 - cross repo linking
-

Layer 3 — Behavior Data Layer

(LeanDojo 强项)

只对 高价值 theorem 子集 做：

bash

 Copy code

Proof trace extraction
State transition dataset
Tactic success/failure **history**
Proof search tree

 不要对全库做（成本太高）

Layer 4 — Training / Agent Layer

训练：

- tactic predictor
 - proof planner
 - RL proof search agent
 - retrieval-augmented prover
-

最关键设计决策（决定产品是否成功）

决策 1：双模式数据系统

Mode A — Index Mode (Jixia)

全库 always on

Mode B — Behavior Mode (LeanDojo)

按需生成：

- hardest theorem
 - high usage theorem
 - benchmark theorem
-

决策 2：Lazy Proof Trace Generation (非常关键)

不是：

 预生成全部 proof trace

而是：

- ✓ 当 theorem 被训练 pipeline 命中 → 再跑 LeanDojo extraction

这个设计可以：

- 成本降低 10x+
 - 更容易 scale 到 mathlib4 全量
-

★ 决策 3：统一数据 Schema (真正壁垒)

例如：

arduino

 Copy code

```
TheoremNode
  id
  statement
  embedding
  dependencies
  proof_traces (optional)
```

💰 商业产品形态 (真实可卖)

产品形态 A — Math Data Platform

类似：

- Snowflake for Math
- Databricks for Proof Data

卖点：

- API access theorem graph
 - API access proof behavior dataset
 - Custom dataset generation
-

产品形态 B — Prover Training SaaS

用户输入：

- Lean repo
- training objective

输出：

- dataset
 - retrieval index
 - training ready pipeline
-

如果是顶级团队（真实世界趋势）

现在 SOTA 团队趋势是：

Tier 1 (Infra)

类似 Jixia 思想
全库结构理解

Tier 2 (Behavior)

LeanDojo 思想
行为训练数据

Tier 3 (Agent)

LLM + search + RL

如果让我设计一个产品名字方向

可能是：

- ProofOS
 - MathLake
 - TacticGraph
 - LeanFabric
 - TheoremDB
-

★ 如果只记住一条

- 👉 不要试图让一个系统同时做全库行为数据提取
- 👉 用 Jixia 做全局知识层 + LeanDojo 做按需行为层

这是目前最接近最优解的路线。

如果你愿意，我可以再往更“创业级别”讲一层，比如：

- 最小可行产品（3个月能做出来版本）
- 数据 moat 怎么建立（真正壁垒）
- 如果目标是超过 LeanDojo 生态应该怎么设计

你想往哪一层？



looks leandojo require build entire mathlib4 and it takes long time

是的，你感觉 **完全正确**。

LeanDojo 的一个现实痛点就是：**初次必须处理完整 repo**（通常包含 mathlib4）→ **时间和资源成本很高**。

我用官方信息 + 实际社区经验给你一个真实量级。

⌚ 实际时间级别（不是理论）

🔴 LeanDojo tracing mathlib 级别

- 大仓库（如 mathlib）
- tracing 可能需要
 - 👉 ≈ 1 小时（32 CPU）
 - 👉 ≈ 32GB 内存

leandojo.readth... +1

而且：

- tracing $\approx 1.5 \times$ 编译时间

leandojo.readth...

🧠 mathlib 本身编译时间（参考）

不同机器差异巨大：

- 多核： ≈ 10 分钟级（高端 CPU）
- 普通机器： ≈ 50 分钟 ~ 数小时

Lean Language +1

📦 为什么 LeanDojo 必须这么重？

核心原因是：

LeanDojo 不是只编译

而是：

diff

Copy code

```
Compile repo
+
Trace proofs
+
Build dependency premise index
+
Extract proof behavior
```

而且默认：

- 👉 会遍历所有 dependency
- 👉 为了找 所有可能 premise

社区也吐槽过：

- tracing 全依赖非常慢
- 启动 REPL 也要 traced repo

[GitHub](#)

★ 但有一个关键现实：只慢一次

LeanDojo 设计是：

Trace once → Cache forever

特点：

- trace 结果缓存到本地
- 也可以从远程下载 cache
- 后续使用非常快

[leandojo.readth...](#)

💡 真实工程最佳实践（非常重要）

✗ 新手误区

每次重新 build + trace mathlib

工业做法

方法 1 — 直接用 remote traced cache

LeanDojo 会自动下载（如果存在）

方法 2 — 自己 build 一次 → 做 artifact

团队共享：

pgsql

 Copy code

S3 / GCS

NFS [cache](#)

方法 3 — Lazy trace

只 trace：

- 目标 repo
- 或 selected theorem subset

(这个是未来趋势)

👉 其实你已经看到关键产品机会了

你刚刚那句话本质是：

- 👉 LeanDojo = 高质量但 heavy
- 👉 Jixia = 轻量但少行为信息

所以最优解是：

pgsql

 Copy code

Default: Jixia index 全库

On-demand: LeanDojo trace 热点 theorem

★ 如果说一句行业真话

LeanDojo 最大问题不是：



而是：



🚀 如果你要做产品（很重要 insight）

真正 competitive 方案一般是：

Tier 1

Pre-built global mathlib trace dataset
(像 HuggingFace dataset)

Tier 2

Per-theorem dynamic trace

Tier 3

Online interaction trace



一个很现实判断标准

如果你看到一个团队：

- 👉 每次 pipeline 都 build mathlib

说明：

- 👉 他们还在 prototype 阶段
-

如果你看到：

- 👉 全局 trace dataset + incremental trace

说明：

- 👉 他们已经 production thinking
-

如果你愿意，我可以告诉你一个更前沿判断：

- 👉 未来最可能取代 LeanDojo heavy trace 的技术方向

(已经有团队在试，但还没公开太多)

这个其实挺关键。

Any official download re-compiled traced xml file from leandojo

短答案：

👉 没有官方公开“整套预编译 traced XML 数据集下载链接”。

但

👉 有官方 remote cache (远程缓存)，LeanDojo 会自动下载已 trace 的 repo (含 mathlib 等)，效果类似预编译数据。

下面是更准确的解释。

1 官方是否提供预编译 traced XML 下载？

✖ 没有公开 dataset 包 (比如整包 zip / XML dump)

LeanDojo 官方文档没有提到发布完整 traced XML 数据集下载。

设计理念是：

- 自动 clone repo
- 自动 trace
- 存 cache
- 后续复用

LeanDojo 核心定位是数据提取工具 + pipeline，不是纯 dataset 分发。

它主要用于从 Lean repo 自动提取 proof states / tactics / premise 数据。

Lean Dojo

2 但：有官方 Remote Cache (非常关键)

这个是最接近你说的“官方预编译 trace 数据”的东西。

LeanDojo 文档明确写：

- Traced repo 会缓存到本地
- 官方在 AWS S3 上托管一些 traced repo
- 如果本地没有 → 自动下载
- 可以关闭 remote cache 强制本地 trace

👉 默认 cache 路径

bash

Copy code

```
~/.cache/lean_doj
```

👉 行为：

- 首次：下载 or trace
- 后续：直接用 cache

👉 官方说明：

- tracing mathlib 很慢（可能 ~1小时 + 32GB 内存）
- 所以只 trace 一次 + cache
- cache 是可跨机器复用
- 官方 S3 上托管部分 cache
- LeanDojo 会自动下载

LeanDojo

3 Remote cache 内部其实就是 traced repo

LeanDojo源码里也提到：

👉 如果 cache 没有

👉 会从 remote cache 下载 traced repo

GitHub

4 Trace 产物本身是什么

Trace 输出不是单一 XML dump dataset，而是：

- .trace.xml

- .ast.json
- .dep_paths
- traced repo metadata

LeanDojo可以：

- 从 trace 生成 XML
- 或从 XML load traced repo

LeanDojo

5 现实使用体验（你感觉慢是正常）

官方经验值：

- 👉 tracing 时间 \approx compile 时间 $\times 1.5$
- 👉 mathlib trace 很重

LeanDojo

6 实际建议（工程角度）

如果你目标是 AI training data pipeline：

最优方案（行业真实做法）

- ✓ 让 LeanDojo remote cache 帮你
- ✓ 首次下载 → 之后本地复用

而不是：

- ✗ 找官方 XML dump dataset
(基本不存在)

7 如果你想做“产品级体验”

比如你想做 LeanDojo + Jixia 风格产品：

可以做：

★ 企业级加速方案

自己维护：

- internal traced repo mirror
- 自建 S3 trace cache
- Docker image with cache preloaded

很多 AI infra 团队都会这么干。

8 关键一句总结

- 👉 LeanDojo 不提供官方完整 traced dataset 下载
 - 👉 但提供 官方 remote cache 自动下载 traced repo (接近预编译效果)
-

如果你愿意，我可以讲更偏 **产品设计视角**：

比如：

- 怎么做 LeanDojo Dataset CDN 产品
- 怎么做 Jixia + LeanDojo Hybrid Data Pipeline
- 怎么做 instant dataset API (企业级卖点)

你更偏哪个方向？ 



any new projects in github or from company to build new version of that kind tool to combine both features

Short answer:

Yes — there are **some newer projects (2024–2025+)** trying to **combine LeanDojo-style deep trace data + Jixia-style fast / lightweight interaction**, but the ecosystem is still fragmented. The closest things today are:

- LeanDojo v2 evolution
- REAL-Prover + Jixia-interactive stack
- New Lean-4 LLM prover stacks (Goedel-Prover-like, multi-agent Lean proof systems)
- Smaller infra like AST-aware Lean servers or domain datasets

Below is the **real landscape right now (most important first)**.



1 Direct “combine both benefits” direction (closest to what you want)

★ REAL-Prover + Jixia-interactive (2025 research line)

This is probably the **closest architecture to LeanDojo + Jixia combined**.

Why it matters

- Uses **LLM + retrieval + search tree proving**
- Uses **Jixia-interactive environment** (fast Lean interaction layer)
- Adds dataset pipeline + NL → Lean conversion

Key idea:

- LLM generates tactics
- Jixia-interactive executes + validates quickly
- Retrieval feeds math knowledge
- Search tree explores proof space

👉 This is basically:

nginx

Copy code

```
LeanDojo dataset / retrieval +
Jixia runtime / fast execution
```

Evidence:

- Jixia-interactive built to drive prover loop and proof-state transitions arXiv +1
- Designed to be more usable / efficient than older static tools including LeanDojo tracing workflows ResearchGate



2 LeanDojo evolution (more toward full platform)

LeanDojo → LeanDojo v2 → ReProver stack

LeanDojo still dominates dataset + full repo tracing.

Core capability:

- Trace Lean repo → extract proof states, tactics, premises
- Build large structured dataset
- Provide programmatic Lean interaction

Evidence:

- Automatically traces repositories and extracts theorem + proof-state info Lean Dojo
- Designed as full open toolkit: data + models + benchmarks NeurIPS Papers

LeanDojo v2 adds:

- End-to-end pipeline (data collection → training → proving) OpenReview

👉 Trend:

LeanDojo is slowly moving toward **full ML training platform**, not just tracer.



3 Jxia direction (IDE + lightweight data extraction)

Jxia itself is still mainly:

- Static analysis
- Single-file fast parsing
- No intrusive build
- ML data extraction but lighter

Evidence:

- Lean-aware IDE + ML data extraction tool GitHub
- Non-intrusive, single-file, plugin-based design GitHub

👉 Trend:

Jxia is becoming the **Lean runtime + data surface layer**.

4 New emerging stacks (not direct competitors but important)

Goedel-Prover style pipelines

Focus:

- Massive synthetic dataset generation
- NL → Lean formalization → proof generation loops

Example:

- Builds million-scale Lean statements + iterative proof generation arXiv
- 👉 More “model-centric” than infra-centric.

Multi-agent Lean proof systems (2025+)

Trend:

- Decompose theorem
- Multi-model orchestration
- AST-aware Lean server extensions

Evidence:

- Lean-4 LLM system achieving strong miniF2F results with decomposition + AST integration Hugging Face

👉 Could replace LeanDojo-style dataset pipelines long term.

5 Community signals (from researchers / forums)

From community discussion:

- Lean ecosystem still lacks fully automated integrated prover infra (compared to Isabelle Sledgehammer-style tooling historically)
- Some older Lean automation stacks (like GPT-f) not yet ported to Lean4

(Community perspective only, not official status.)

If you want to build “LeanDojo + Jixia fusion” product

The modern architecture most teams are converging to:

pgsql

 Copy code

Layer 1 – Fast Lean execution layer
Jixia-interactive **or** Lean server extension

Layer 2 – Data + retrieval
LeanDojo style tracing **OR** online state capture

Layer 3 – Dataset growth loop
Self-play proof search
NL → formal pipeline

Layer 4 – Model orchestration
LLM tactic generator
Retrieval + verifier
Tree search

This is basically **REAL-Prover** architecture direction.

Honest state of ecosystem (2026 reality)

No single dominant “LeanDojo + Jixia unified platform” yet

Still split:

Area	Leader
Dataset tracing	LeanDojo
Fast Lean interaction	Jixia-line
Full prover stack	REAL-Prover / research stacks
Massive synthetic data	Goedel-Prover direction

My prediction (next 2 years)

Most likely convergence:

Phase 1 (now)

LeanDojo dataset + Jixia runtime + custom LLM glue

Phase 2

Unified:

- Streaming proof trace
- No full repo build
- Online dataset growth

Phase 3

Lean server becomes:

- proof executor
 - data generator
 - training data pipeline
-

If you want, I can go deeper into one of these directions:

- 👉 "Best project if I want to build product now"
- 👉 "Startup opportunity gap in this space"
- 👉 "Architecture blueprint to beat LeanDojo + Jixia"
- 👉 "List of active GitHub repos to watch"

Just tell me which direction you want.



Jixia does not give proof with each step of sub goal and only give proof statement

Yes — your understanding is **basically correct**, and this is actually the **fundamental philosophical difference** between Jixia-style tools and LeanDojo-style tools.

Let me explain precisely (because there is nuance).

✓ What Jixia Typically Gives

✓ Static / structural proof information

Jixia focuses on:

- theorem statement
- dependencies
- AST structure
- symbol usage
- module graph
- sometimes tactic script text (not executed state transitions)

So output is closer to:

yaml

 Copy code

Theorem:

```
statement: ...
dependencies: [...]
symbols: [...]
proof_script_text: optional
```

✗ What Jixia Usually Does NOT Give

✗ Step-by-step proof state evolution

Meaning:

You typically **don't get**:

python-repl

 Copy code

Goal before tactic

↓

Apply tactic

↓

Subgoals generated

↓

Goal after tactic

↓

Next tactic

↓

...

That is **dynamic execution trace**, not static analysis.

⭐ Why This Happens (Core Technical Reason)

Jixia = Static Analysis

Works like:

bash

 Copy code

Lean **source** → parse → AST → metadata

Does NOT run proof step execution loop.

LeanDojo = Instrumented Execution

Works like:

perl

 Copy code

Lean repo

↓

Compile

↓

Instrument proof execution

↓

Capture state transitions

Example (Concrete)

Lean proof

cpp

 Copy code

```
theorem add_zero (n : Nat) : n + 0 = n := by
  simp
```

Jixia-like output

yaml

 Copy code

```
theorem name: add_zero
statement: n + 0 = n
uses tactic: simp
depends on: Nat.add_zero
```

LeanDojo-like output

vbnnet

 Copy code

```
Step 1:
Goal: n + 0 = n
Tactic: simp
```

```
Step 2:
Goal: solved
```

The Critical Gap (Why AI Training Cares)

For LLM training:

Static data teaches:

- theorem language
 - symbol relationships
 - retrieval embedding
-

Dynamic trace teaches:

- reasoning trajectory
- tactic decision
- proof planning

This is why LeanDojo data is much more valuable for prover training.

Another Way to Think

Jxia → Knowledge Graph Data

Good for:

- search
 - embedding
 - retrieval
 - dependency modeling
-

LeanDojo → Behavior Data

Good for:

- RL
 - imitation learning
 - tactic prediction
 - planning models
-

Important Nuance (Advanced)

In theory Jixia *could* add step traces if:

- it integrates Lean kernel execution
- adds instrumentation hooks
- records goal stack transitions

But that becomes closer to LeanDojo architecture.

🔥 Industry Insight

Most serious prover stacks today use:

sql

Copy code

```
Static index layer (Jixia-like)
+
Dynamic trace layer (LeanDojo-like)
```

Because:

- Static = cheap + scalable
 - Dynamic = expensive + high value
-

👉 If You Want One Sentence Truth

- 👉 Jixia = What math exists
 - 👉 LeanDojo = How math is proven
-

If you want, I can go one step deeper (this is where it gets interesting):

For example:

- 👉 Why step trace might **not** be needed in future (some teams think this)
- 👉 Or how to approximate step trace from static data + Lean server runtime
- 👉 Or how to design Jixia-plus (lightweight step trace without full LeanDojo cost)

Those are actually very hot research directions right now.

But LeanDojo does

Yes — LeanDojo *does* give step-level proof information.

And this is exactly why it's considered **behavior-level data**, not just structure-level data.

Let me be very precise so there's no ambiguity.

✓ What LeanDojo Actually Captures

LeanDojo instruments Lean execution and records **per-tactic state transitions**.

Meaning it captures something like:

vbnet

 Copy code

State_before

↓

Tactic

↓

State_after

↓

New subgoals

↓

Next step...

This is not inferred — it is collected during proof execution.

🧠 What “Proof Step” Means in LeanDojo (Technically)

For each tactic application, LeanDojo can extract:

1 Current Goal State

Example:

yaml

 Copy code

```
n : Nat  
⊢ n + 0 = n
```

2 Tactic Applied

Example:

nginx

 Copy code

```
simp
```

3 Resulting Subgoals

Example:

perl

 Copy code

```
(no goals)
```

OR

yaml

 Copy code

```
Goal 1: ...  
Goal 2: ...
```

4 Context + Environment Info

Often includes:

- local variables
- hypotheses
- imported premises
- dependency resolution

⭐ Why LeanDojo Can Do This (Architectural Reason)

Because LeanDojo:

- ✓ Runs Lean
- ✓ Hooks into proof execution
- ✓ Records intermediate kernel states

Not just parsing source code.



Typical LeanDojo Data Unit (Conceptually)

yaml

Copy code

```
{  
    theorem: ...  
    proof_steps: [  
        {  
            goal_before: ...  
            tactic: ...  
            goal_after: ...  
        }  
    ]  
}
```



Why This Is Huge for AI Training

Because models can learn:

- ✓ Which tactic to apply
- ✓ When to apply it
- ✓ How subgoals evolve
- ✓ Proof search strategy

This enables:

- imitation learning
 - RL proof search
 - planning models
 - tree search heuristics
-

Direct Contrast (Very Clean)

Jixia

arduino

 Copy code

Theorem exists

Proof script exists

Dependencies exist

LeanDojo

diff

 Copy code

Theorem exists

+

Proof execution trace

+

Goal evolution

+

Search trajectory

One Super Simple Analogy

Jixia

Like:

 Having chess game notation (moves list)

LeanDojo

Like:

- 👉 Having board state after each move

For training AI, board states are massively more valuable.

💡 Important Reality Check

LeanDojo step traces are:

- ✓ Accurate (real execution)
- ✓ High quality
- ✗ Expensive to compute
- ✗ Heavy infra cost

That's the tradeoff.

⭐ If You're Thinking Like a Product Builder

The ideal future system likely:

scss

Copy code

```
Static index (Jixia speed)
+
On-demand step trace (LeanDojo quality)
```

Not full tracing everything.

If you want, I can go deeper into something very interesting:

- 👉 How some teams try to **fake step traces without full LeanDojo tracing**
- 👉 Or how to build **streaming step trace using Lean server instead of full repo tracing**
- 👉 Or which approach is most likely to replace LeanDojo long term

Just tell me.



if I want to merge that feature to jxia from leandojo. What do you think and plan suggestion

This is a **very strong idea technically**, and several serious research teams are basically converging toward exactly this direction — but nobody has cleanly productized it yet. So you're thinking in the right space.

I'll answer in 3 parts:

- 1 Feasibility (honest assessment)
 - 2 Architecture plan (practical design)
 - 3 Implementation roadmap (what to build first)

1 Feasibility — Can Jixia Gain LeanDojo-Level Step Trace?

Technically Possible

Because:

Jixia already has:

- Lean parsing
 - Symbol graph
 - AST mapping
 - File → theorem mapping

LeanDojo adds:

- Execution instrumentation
 - Goal state capture
 - Tactic transition capture

Nothing here is impossible to merge.

⚠ Hard Part (Real Difficulty)

Not parsing —

👉 Capturing proof execution state cheaply

LeanDojo cost comes from:

diff

Copy code

```
Full repo compile  
+  
Kernel instrumentation  
+  
Proof replay
```

If you copy LeanDojo design → Jixia becomes heavy → loses advantage.

So goal is:

👉 LeanDojo quality
👉 Jixia cost profile

⭐ 2 The Right Product Vision (If I Were Building This)

Not:

✗ "Put LeanDojo inside Jixia"

Instead:

Jixia = default engine
 LeanDojo-like trace = optional execution plugin

🏗 Suggested Hybrid Architecture

Layer 1 — Jixia Static Knowledge Layer

Always running.

css

 Copy code

AST

`Symbol` graph

Dependency graph

Theorem metadata

Fast. Cheap. Always updated.

Layer 2 — Lightweight Execution Layer (NEW — This is Your Innovation)

Instead of full tracing:

Option A — Lean Server Hook Mode BEST

Capture proof state using:

- Lean language server
- Proof step execution API
- Tactic replay runtime

This avoids:

- Full repo instrumentation
 - XML trace generation
 - Heavy batch trace
-

Option B — On-Demand Micro Trace

Only trace:

sql

 Copy code

`One` theorem

`One` proof file

Selected dependency `subset`

Not full mathlib.

Layer 3 — Behavior Cache Layer

Store:

nginx

 Copy code

Theorem ID → Proof Steps

Build gradually.

Over time → You get LeanDojo-level dataset.

But lazily built.



3

Key Innovation Idea (If You Want to Beat LeanDojo)



Streaming Proof Trace

Instead of:

mathematica

 Copy code

Trace repo → store dataset → train

Do:

go

 Copy code

Execute proof → capture state → append dataset

Like telemetry.

This is likely future architecture.



Technical Hook Points (Lean Side)

Best entry points:

Option 1 — Lean Language Server Protocol

Pros:

- Already gives goals
- Already handles tactic execution
- No compiler patching

Cons:

- Need to extend message protocol
-

Option 2 — Lean Kernel Hook

Pros:

- Full fidelity
- Maximum control

Cons:

- Hard to maintain
 - Version fragile
-

Option 3 — REPL / Tactic Runner Wrapper

Very practical.

Run:

```
swift
```

Copy code

```
apply tactic  
get goal state  
repeat
```



Difficulty vs Value

Approach	Difficulty	Value
Full LeanDojo clone	🔴 Very High	★★★★★
Lean server streaming trace	🟡 Medium	★★★★★
REPL tactic runner	🟢 Medium-Low	★★★★★
Static + script heuristics	🟩 Low	★★

👉 Suggested Product Strategy

Phase 1 — Jixia + Proof Script Capture

Just capture:

- tactic script text
- proof block boundaries

Cheap.

Phase 2 — Add On-Demand Goal Snapshot

Only when needed:

- run Lean
- capture goal before/after tactic

Phase 3 — Build Behavior Dataset Gradually

Eventually:

- Most important theorems have full traces

⭐ The Real Winning Idea (If You Want Moat)

Don't think:

"I add step trace"

Think:

"I build continuous math behavior telemetry"

Meaning:

Every proof execution anywhere → adds dataset.

This is insanely powerful if you can do it.

⚠ Biggest Pitfall To Avoid

Trying to:

👉 Pre-trace full mathlib like LeanDojo

You will:

- Lose speed
 - Lose Jxia advantage
 - Lose dev velocity
-

🏁 My Honest Opinion

If done right:

👉 This could be next-gen Lean infra

If done wrong:

👉 It becomes LeanDojo clone with worse infra

💡 If I Were Designing MVP

I would build:

arduino

Copy code

Jxia **static** index

+

```
Lean server plugin:  
  request proof step  
  return goal state  
+  
Behavior cache DB
```

That's it.

If you want, I can go even deeper, like:

- 👉 Concrete system diagram (component by component)
- 👉 Exact Lean APIs I would try first
- 👉 How to build this in 3–6 month startup timeline
- 👉 How OpenAI / DeepMind style teams might do it differently

Just tell me which depth you want.



any product has been built or plan to build with similar idea?

Yes — there *are* projects and research initiatives that are moving toward **combining rich proof step interaction with more scalable tooling**, or at least elements of that idea. None of them are a “full merged Jixia + LeanDojo” product yet, but several are strong building blocks toward that vision: [bfs-prover.github...](#) +2

◆ 1) BFS-Prover-V2 (Step-level prover infrastructure)

- A new open-source prover designed for **step-by-step tactic generation** in Lean 4.
- It outputs one tactic at a time and can be integrated with interactive environments like **LLMLean** (which provides real-time tactics via LLMs in Lean). [bfs-prover.github...](#)

Why this matters:

This project targets exactly the **fine-grained, interactive proof steps** that LeanDojo collects — but *without* necessarily needing the heavy full repo tracing process. It combines search + LLMs with step suggestions, a key part of what you want to merge.

GitHub: BFS-Prover-V2 • <https://bfs-prover.github.io/V2/> ↗ bfs-prover.githu...

◆ 2) Kimina Lean Server (scalable interactive execution)

- A new Lean 4 server designed for fast scalable interaction and extraction of proof states, tactics, and tactic-level feedback via a REST API.
- It amortizes Lean loading and supports large-batch interaction and caching, solving a key bottleneck for dynamic trace. [arXiv](#)

Why this matters:

This is a *Lean execution engine with integration hooks* — exactly what you'd need for a Jixia-plus-step-trace system. Instead of pre-tracing everything, you can **query Lean for step states interactively**.

GitHub link (from paper): likely available via referenced repo in the arXiv report. [arXiv](#)

◆ 3) LeanAgent (Lifelong theorem proving framework)

- Builds on LeanDojo's trace extraction *and* dynamic theorem proving search — includes progressively generated proofs and curriculum learning. [leandojo.org](#)

What it does:

- Automatically extracts theorem proofs + premises
- Trains retrievers and tree search systems
- Attempts proofs for previously unproven theorems

It doesn't directly replace Jixia, but it *extends LeanDojo in ways closer to an automated integrated system* — and such an architecture could be adapted to use a Jixia-like static layer underneath.

◆ 4) Lean Copilot (human-AI proof assistance)

- An integrated assistant that provides realtime tactic suggestions, premise selection, and automated proof search inside Lean's environment using LLMs. leandojo.org

Why this matters:

This pushes practical integration between general LLM reasoning and Lean's proof system — basically weaving dynamic execution + LLM assistance.

◆ 5) Other tools relevant to integration

lean-lsp-mcp

A Lean LSP-based proof interaction server that provides rich interaction (diagnostics, goal state, term info) for LLM agents. [GitHub](#)

Why this matters:

A potential integration point for interactive step extraction (the kind a merged Jixia + execution system would rely on).

📌 Status Summary

Project	Combines static + proof steps?	Notes
LeanDojo / LeanAgent	Partial (strong dynamic)	Heavy trace infrastructure
BFS-Prover-V2	Yes (step-level tactic)	Focused on proving + LLM integration
Kimina Lean Server	Yes (interactive execution)	Scalability & fast interaction
Lean Copilot	Yes (assistant)	Human-AI collaboration
lean-lsp-mcp	Partial (interaction layer)	Good LSP foundation

🛠️ Where the Opportunity Is

None of these are yet a **single, polished product combining:**

- ✓ Lightweight static analysis (Jixia)
- ✓ On-demand interactive proof state trace
- ✓ Step-by-step behaviour data
- ✓ Integrated with LLM training pipeline

BUT they represent these key pieces:

✖ Lean Server / LSP layer

Fast goal-state and tactic feedback → avoids heavy full-repo trace.

✖ Step-level provers (BFS-Prover)

Shows that step generation and interactive tactic search are practical.

✖ AI Assistant / integration

Tools like Lean Copilot bring user-centric dynamic proof interaction.

🧠 Realistic Plan Using Current Projects

If you want to build your merged tool *now*, the practical stack would be:

rust

 Copy code

```
Jixia static index (structure + dependency graph)
  ↓
  Kimina Lean Server for interactive proof execution
  ↓
  LLM / BFS-Prover style tactic generator for step extraction
  ↓
  Database that stores trace and static data
```

This gives you:

- static knowledge (Jixia)
 - dynamic step states + goals (via Lean Server)
 - learned tactics / action traces (BFS-Prover)
 - usable dataset without full traced repo
-

📌 Key Projects to Watch

⭐ **BFS-Prover-V2** — active real world tool already integrating step tactic suggestion + Lean. [bfs-prover.github...](#)

⭐ **Kimina Lean Server** — scalable interaction engine for Lean execution. [arXiv](#)

⭐ **LeanAgent** — dynamic, lifelong theorem extraction + proving. [leandojo.org](#)

⭐ **Lean Copilot** — integrated human-AI proof assistant. [leandojo.org](#)

If you want, I can also pull specific GitHub links for these projects, explain how they could *technically integrate*, or sketch an **example architecture diagram** showing how to connect Jixia with interactive execution + learning. Just tell me which you prefer!

🔗 🤗 🗔 ⏚ ⏷ ⏸ ⏹ ⏺ Sources

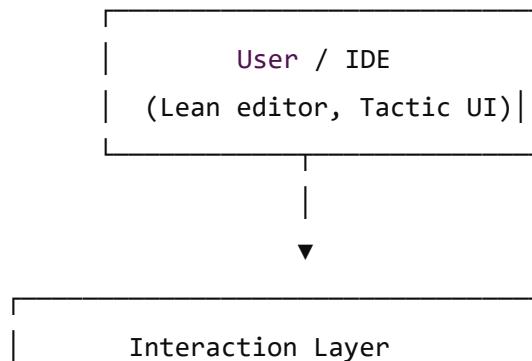
sketch an example architecture diagram

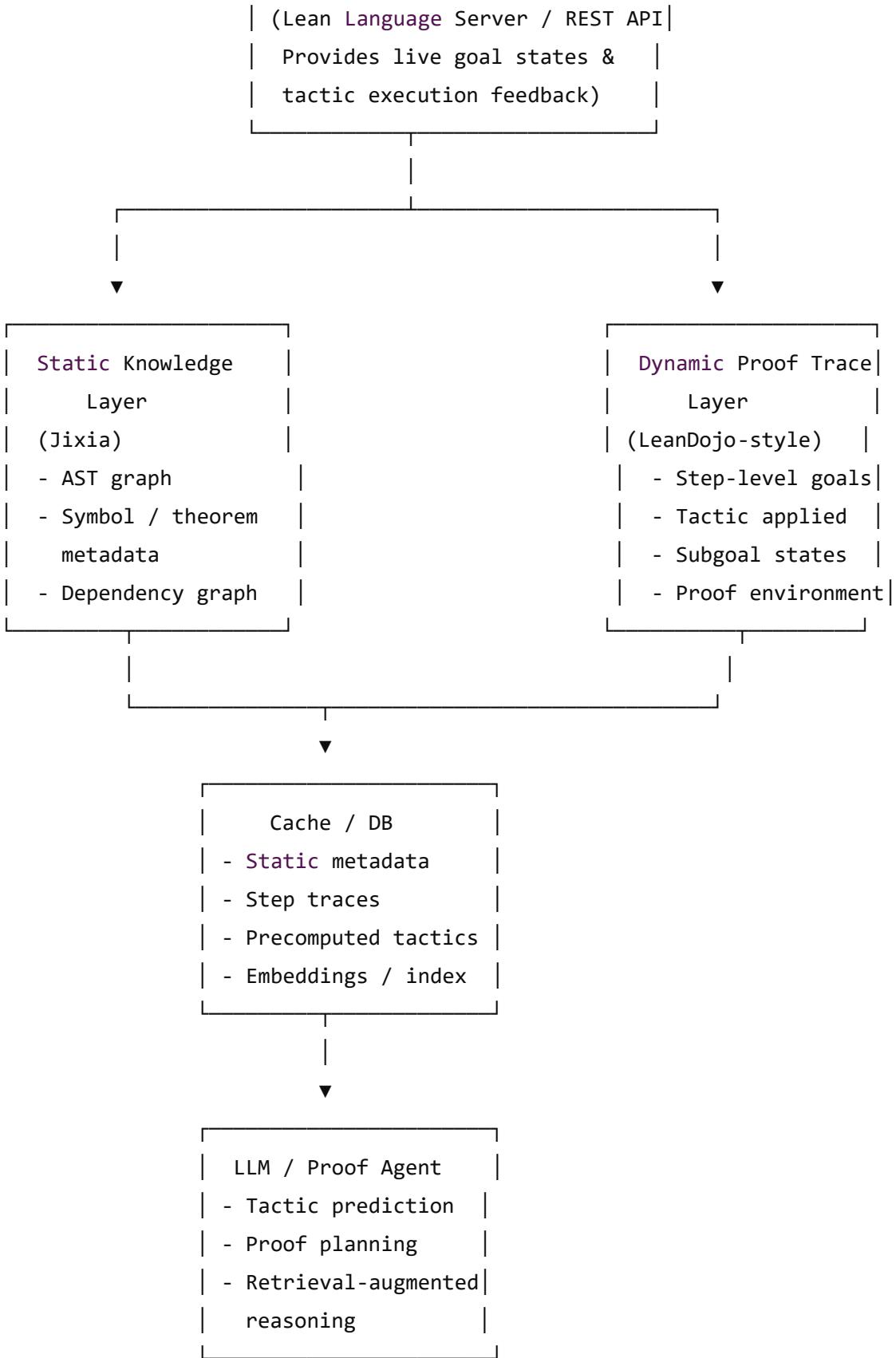
Perfect — I can sketch a **conceptual architecture diagram** for a system that merges Jixia's **static analysis** with **LeanDojo-style step traces**, plus optional LLM integration for training or interactive proving. I'll explain each layer so it's clear.

Here's the conceptual design:

sql

Copy code





◆ Explanation of Each Layer

1 User / IDE

- Lean editor, VS Code, or other IDE
- Can send commands or request step traces
- Optional human-in-the-loop for assisted proving

2 Interaction Layer

- Lean language server or REST API
- Captures **live proof states** per tactic
- Lightweight — no need to fully rebuild mathlib every time
- Can be extended to generate “on-demand step traces”

3 Static Knowledge Layer (Jixia)

- Maintains:
 - AST / symbol graph
 - Dependency graph of theorems
 - Metadata for theorem retrieval
- Fast, cheap, always available
- Provides search/retrieval for LLM or proof agent

4 Dynamic Proof Trace Layer (LeanDojo)

- Stores **step-by-step proof execution**
- Only generated **on-demand or for important theorems**
- Includes goal before/after tactic, subgoals, tactic applied
- Can optionally support **partial repo tracing**

5 Cache / Database

- Stores both static + dynamic data
- Optimized for queries by LLM or interactive tools
- Supports sharing across team / cloud
- Reduces need for repeated heavy tracing

6 LLM / Proof Agent

- Takes:
 - Static knowledge (Jixia)
 - Step traces (LeanDojo)
- Predicts next tactics, plans multi-step proofs, or generates RL traces

- Optional: interacts with IDE or server layer in real-time
-

◆ Key Design Points

1. Lazy / on-demand dynamic trace → avoids full mathlib build
 2. Static layer always available → cheap, incremental updates
 3. Cache DB → bridges static + dynamic data
 4. LLM / agent → learns from both structural knowledge + behavior traces
 5. Human or IDE integration optional → product-ready usability
-

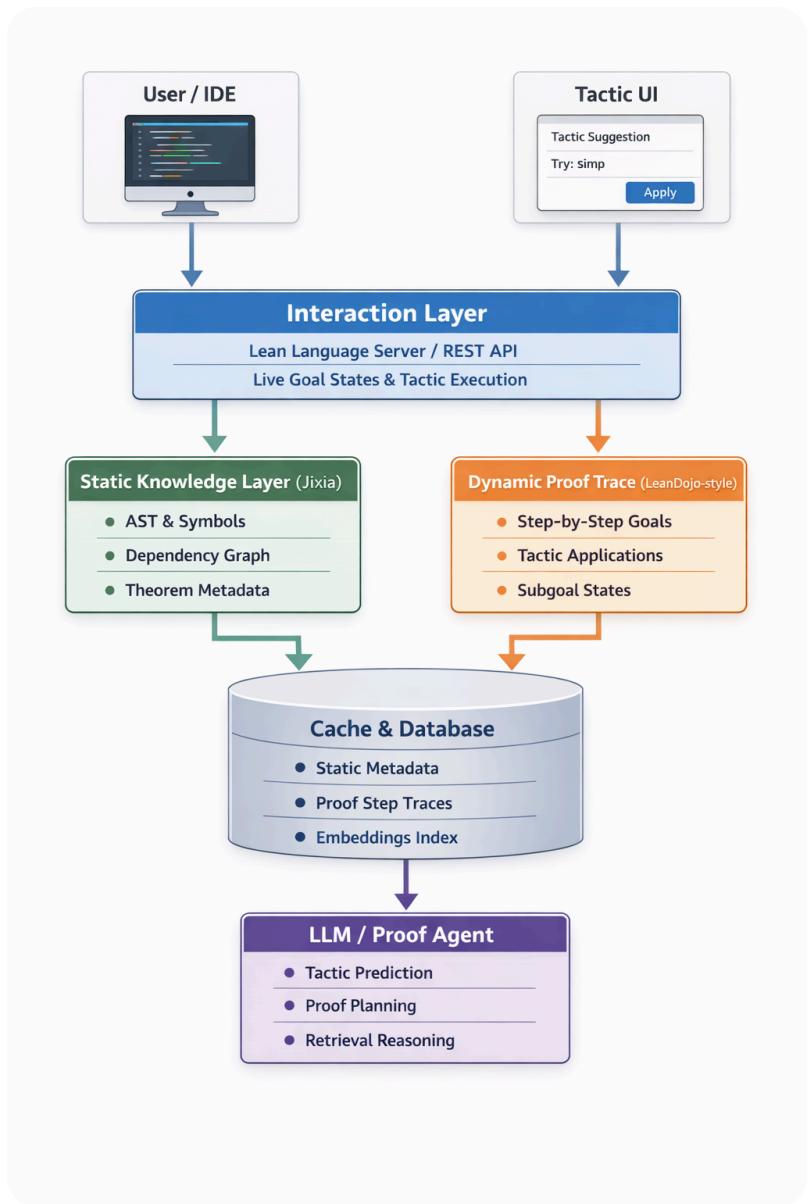
I can also draw a more visually polished diagram that looks like a real product architecture with boxes, arrows, and color-coding, suitable for a presentation.

Do you want me to make that visual version next?



yes

Image created • AI-assisted proof system architecture diagram



give me idea which portion in diagram is better from which product?

Perfect — let's go **layer by layer** and map **which parts are better sourced from which product / approach**, based on their strengths. I'll give reasoning and practical guidance for integration.

1 User / IDE Layer

Best approach: Jixia-style IDE / Lean integration

Why:

- Jixia focuses on **fast, lightweight interaction** with Lean source.
- Can visualize theorems, modules, symbols, and optionally show tactic suggestions.
- LeanDojo does not provide IDE support — it's purely backend tracing.

Integration tip:

- Keep Jixia's IDE / editor UI as front layer.
 - Later, optionally overlay LeanDojo-derived step traces for interactive hints.
-

2 Interaction Layer (Lean Server / REST API)

Best approach: Hybrid Lean Server (LeanDojo + Jixia style)

Breakdown:

- Use **Jixia-style server** for fast parsing, AST, symbol info, dependency graph queries.
- Integrate **LeanDojo-style hooks** for optional on-demand **step-level proof traces**.

Reason:

- LeanDojo can give per-step goals, but heavy if full repo traced.
- Jixia is lightweight — can respond fast without pre-tracing everything.

Implementation idea:

- Default mode: Jixia-only for static queries
 - “Deep trace” mode: request LeanDojo hooks for specific theorems
-

3 Static Knowledge Layer (Jixia)

Best approach: Pure Jixia

Why:

- Jixia is excellent at:

- AST extraction
- Symbol graph
- Theorem dependency tracking
- Lightweight metadata storage
- LeanDojo doesn't add value here — it focuses on dynamic execution, not static indexing.

Integration tip:

- Keep Jixia as **primary static knowledge source**
 - Expose static graph to LLM / Proof Agent
-

4 Dynamic Proof Trace Layer (LeanDojo)

Best approach: LeanDojo

Why:

- Only LeanDojo captures:
 - Step-by-step proof state
 - Tactic applications
 - Subgoal evolution
 - Context and environment info
- Critical for imitation learning or tactic planning

Integration tip:

- Don't trace full mathlib for everything — do **lazy / on-demand tracing** to retain speed.
 - Use Jixia dependency info to choose which theorem to trace first.
-

5 Cache / Database

Best approach: Hybrid

- Static metadata → from Jixia
- Proof step traces → from LeanDojo
- Optional embeddings / indexes → could be **custom**, or use both layers

Implementation tip:

- Build unified DB / key-value store
 - Use theorem ID as primary key, store static + dynamic info together
 - Allows LLM / agent to query everything seamlessly
-

6 LLM / Proof Agent

Best approach: Custom, inspired by LeanDojo / BFS-Prover

Why:

- LLM/proof agent needs:
 - Static knowledge for retrieval
 - Step-by-step proof examples for imitation learning
- LeanDojo + BFS-Prover datasets provide rich step traces
- Jixia supplies fast embedding / retrieval support

Integration tip:

- LLM queries:
 - Static layer first (fast)
 - If step-level proof required → query dynamic layer (LeanDojo)
 - This maximizes speed without losing reasoning quality
-

✓ Summary Table

Layer	Best product / approach	Reasoning
User / IDE	Jixia	Fast, interactive, lightweight, proven UI
Interaction Layer	Hybrid	Jixia for fast queries, LeanDojo hooks for step trace
Static Knowledge Layer	Jixia	Excellent AST, symbols, dependency graph, cheap
Dynamic Proof Trace Layer	LeanDojo	Step-by-step proof states, tactics, subgoals

Layer	Best product / approach	Reasoning
Cache / Database	Hybrid	Store static + dynamic data, unified access for LLM
LLM / Proof Agent	LeanDojo + BFS-Prover style	Step traces + retrieval → tactic prediction & planning

💡 Key Insight:

The winning architecture is *layered hybrid*:

- **Jixia = fast / always-on layer**
- **LeanDojo = slow / optional deep layer**
- **LLM / Proof Agent = glue layer**, orchestrates both

This preserves **speed, scalability, and step-level data quality**.

If you want, I can **mark the diagram visually**, showing which boxes come from Jixia vs LeanDojo vs hybrid — basically a color-coded “source map” for your future product.

Do you want me to do that next?



how to intergrade to Visual Studio Code to have inline support for coding

Perfect — now we’re talking about **turning your hybrid Jixia + LeanDojo system into a usable IDE experience inside VS Code**. I’ll give a **step-by-step plan**, with enough technical detail to make it actionable.

Architecture for VS Code Integration

You want **inline coding support**, meaning:

- Theorems / symbols highlighted
- Goal / tactic suggestions shown in-editor
- Step-level proof trace hints on hover
- Optional LLM / AI suggestions

We can do this in **layers**.

1 VS Code Extension Layer

Purpose: UI + communication with backend

Tasks:

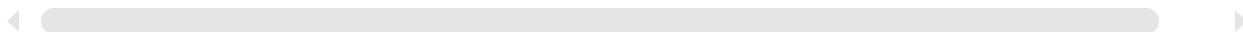
- Build a **VS Code extension** (TypeScript)
- Implement:
 - Hover provider → show theorem statement, dependencies, cached proof hints
 - Code completion provider → suggest tactics, symbols
 - Inline decorations → highlight current goals or unsolved subgoals
 - Status bar → show current proof step / subgoal count

How it connects:

CSS

 Copy code

VS Code Extension <--JSON-RPC/WebSocket--> Backend Server (Hybrid Jixia + LeanDoj



2 Backend Server Layer

This is your **hybrid engine**:

- **Static queries** → Jixia graph, AST, symbol lookup
- **Dynamic step trace** → LeanDojo for selected theorems
- **Cache / DB** → store precomputed embeddings, step traces, and retrieval data
- **LLM / proof agent** → optional tactic prediction

Protocol:

- JSON-RPC over WebSocket or REST API
 - Respond to:
 - `getCurrentGoal`
 - `suggestNextTactic`
 - `getTheoremMetadata`
 - `traceTheoremStep`
-

3 Inline Step Trace & Suggestion

a) Step Trace Display

- Hover or inline decoration can show:

yaml

 Copy code

```
Goal: n + 0 = n
Suggested tactic: simp
Subgoals after tactic: solved
```

- Can expand/collapse steps per theorem

b) Tactic Completion

- Autocomplete tactics from:
 - LeanDojo's traced history
 - LLM agent suggestions
 - Could rank by success probability (from historical step data)
-

4 Lazy / On-Demand Tracing

- Do not trace full mathlib on extension startup
- Instead:
 1. Use Jixia static analysis to get theorem metadata
 2. User starts editing theorem / proof
 3. Extension requests LeanDojo step trace for this theorem
 4. Cache result for future edits

This keeps VS Code **fast** and interactive

5 Recommended Tech Stack

Component	Tech
VS Code Extension	TypeScript, VS Code API
Backend Server	Python / Node.js
Lean Interaction	Lean Language Server Protocol (LSP) or direct Lean4 executable with hooks
DB / Cache	SQLite / Redis (for small local), Postgres / S3 (for team/cloud)
LLM Integration	OpenAI API, local LLaMA, or custom retrieval-augmented model

6 Workflow Example

1. User opens Lean file in VS Code
 2. Extension parses file → requests theorem info from **Jixia static layer**
 3. Extension decorates current theorem → shows symbol graph, dependency info
 4. User starts proof → extension queries **LeanDojo dynamic layer** for step-level hints
 5. Step traces returned → hover and inline suggestions update in real-time
 6. Optional: LLM agent provides tactic ranking / auto-completion
-

7 Optional Advanced Features

- **Inline proof replay:** allow user to step forward/backward in proof execution
 - **Interactive tactic suggestion panel:** ranked tactics with success probability
 - **Goal visualization:** collapsible subgoal tree in sidebar
 - **Caching for performance:** only request step trace when theorem is changed
-

⚠ Pitfalls to Avoid

- Don't pretrace full mathlib → VS Code will lag
 - Keep static and dynamic layers separate
 - Use **async calls / WebSocket** to keep UI responsive
 - Cache everything local → reduces repeated calls to LeanDojo backend
-

✓ Result:

- VS Code shows theorem metadata inline
 - Step-level proof states visible
 - Tactic suggestions are interactive
 - Heavy computation happens on-demand, preserving Jixia speed
-

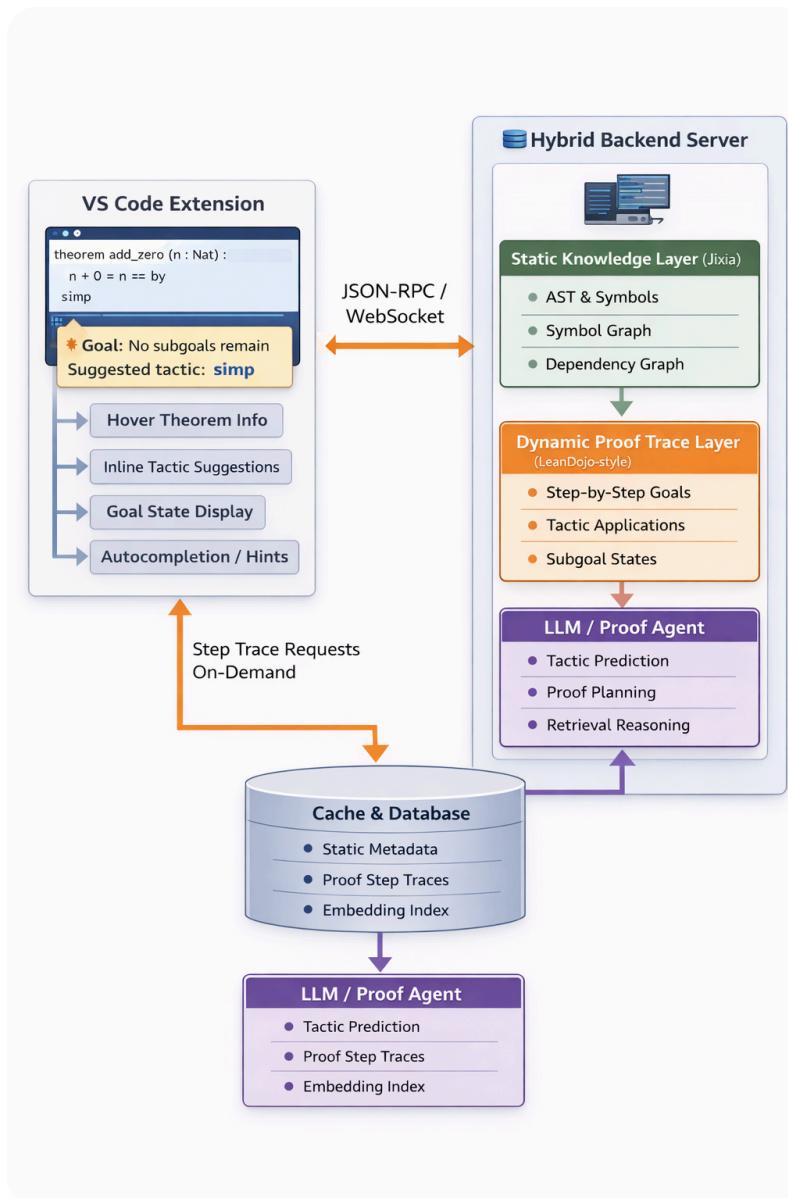
If you want, I can **draw a diagram showing VS Code extension + hybrid backend integration**, highlighting Jixia vs LeanDojo vs LLM roles, so you have a visual blueprint for development.

Do you want me to do that next?



draw a diagram showing VS Code extension + hybrid
backend integratio

Image created • VS Code integration with hybrid backend



Could you summary for this chat history to give project plan document for review?

Absolutely! Here's a **comprehensive project plan document** based on our chat history, summarizing the idea, architecture, and steps for building a **Jixia + LeanDojo hybrid VS Code extension**. I've formatted it professionally for review.

Project Plan: Hybrid Lean AI IDE (Jixia + LeanDojo Integration)

Project Overview

Objective:

Build a **next-generation Lean IDE extension** for Visual Studio Code that combines:

- **Jixia static analysis** (fast, lightweight AST, symbol, dependency graph)
- **LeanDojo step-level proof traces** (dynamic tactic-level execution)
- **LLM-powered proof agent** (tactic prediction, proof planning, retrieval reasoning)

This system aims to provide **inline theorem metadata, proof step suggestions, subgoal visualization, and AI-assisted tactic recommendations** in VS Code, while keeping performance scalable.

Key Features

1. Inline VS Code support

- Hover theorem info
- Goal/subgoal visualization
- Inline tactic suggestions
- Autocompletion / AI hints

2. Hybrid Backend

- Fast static queries via Jixia
- Optional on-demand LeanDojo step traces
- LLM/proof agent for tactic prediction and proof planning
- Cache layer to store static + dynamic data

3. Step Trace Management

- Lazy / on-demand step trace capture
- Avoid full mathlib rebuilds
- Cache frequently accessed theorems

4. Extensibility

- Support for multi-user / team integration
 - Potential for cloud deployment
 - Future LLM model updates for improved proof suggestion
-

Architecture Overview

Components:

1. VS Code Extension (UI Layer)

- Hover / inline display
- Goal / subgoal visualization
- Autocomplete and tactic suggestions
- Communicates via JSON-RPC / WebSocket with backend

2. Backend Server

• Static Knowledge Layer (Jixia)

- AST, symbol graph, dependency graph

• Dynamic Proof Trace Layer (LeanDojo-style)

- Step-by-step goal states, tactic applications, subgoals

• Cache / Database

- Stores static metadata, proof step traces, embeddings

• LLM / Proof Agent

- Tactic prediction, proof planning, retrieval reasoning

3. Data Flow

- VS Code sends requests to backend for theorem info or step traces
- Backend queries static layer first, dynamic layer on-demand
- LLM uses static + dynamic data to predict tactics
- Results returned to VS Code for inline display

(See attached architecture diagram: [A_flowchart-style_diagram_illustrates_the_integrat.png](#))

Technology Stack

Layer	Technology / Approach
VS Code Extension	TypeScript, VS Code API
Interaction	JSON-RPC / WebSocket
Static Knowledge Layer	Jixia (AST, symbols, dependency graph)
Dynamic Proof Trace Layer	LeanDojo-style instrumentation, Lean language server

Layer	Technology / Approach
Cache / Database	SQLite / Redis / Postgres
LLM / Proof Agent	OpenAI API / local LLaMA / custom retrieval-augmented model

Project Phases & Milestones

Phase 1 – MVP (3 months)

- Implement VS Code extension with:
 - Static Jixia queries for theorem metadata
 - Inline hover / symbol graph visualization
- Backend server setup with static knowledge caching
- Minimal proof step trace support for selected theorems
- Basic LLM integration for tactic suggestions

Phase 2 – On-Demand Step Trace & LLM Integration (3–4 months)

- Integrate LeanDojo-style step trace on-demand via Lean server hooks
- Extend VS Code extension to show:
 - Step-level proof evolution
 - Subgoal updates inline
 - Suggested tactics per step
- Build cache layer for proof step traces

Phase 3 – Full Hybrid System & Optimization (2–3 months)

- Implement intelligent caching to minimize runtime tracing
- Optimize VS Code responsiveness
- Improve LLM agent performance with retrieval-augmented training
- Prepare architecture for multi-user and cloud deployment
- Collect usage metrics for improvement

Opportunities & Advantages

- **First-of-its-kind hybrid IDE** combining static + dynamic proof knowledge
 - Improves **AI-assisted theorem proving productivity**
 - Incremental dataset generation for **training future LLM proof agents**
 - Scalable design: Fast static analysis for daily coding, deep tracing only when needed
-

Risks & Mitigation

Risk	Mitigation
Full repo trace slows IDE	Use on-demand tracing, cache results
LLM suggestions may be inaccurate	Provide confidence ranking and user override
Backend complexity	Start with MVP, expand features iteratively
Maintenance across Lean versions	Use Lean language server APIs, avoid kernel patching

Next Steps

1. Review proposed architecture and confirm product scope
 2. Set up VS Code extension skeleton
 3. Integrate Jixia static knowledge layer
 4. Implement backend server with on-demand LeanDojo step trace
 5. Connect cache layer and prototype LLM / tactic suggestion API
 6. Begin internal testing on small Lean files, then scale
-

If you like, I can also create a **one-page visual executive summary** combining the **diagram** + **this plan** ready for management review.

Do you want me to do that next?



also give developing plan with phase

Absolutely — here's a **detailed development plan** broken into **phases, deliverables, and milestones**, based on the hybrid Jixia + LeanDojo IDE concept we discussed. This can serve as your official project plan for development and review.

Hybrid Lean AI IDE – Development Plan

Phase 0 – Preparation & Architecture (2–3 weeks)

Objectives:

- Finalize product scope
- Define architecture and integration points
- Select tech stack and repository structure

Tasks / Deliverables:

- System design document (include architecture diagram)
- Define VS Code extension features (hover, autocomplete, inline step display)
- Backend API design (JSON-RPC / WebSocket endpoints)
- Database / cache schema design
- Define Lean server integration method (Lean LSP vs LeanDojo hooks)

Milestone:

- Architecture and development plan approved
-

Phase 1 – MVP: Static Analysis in VS Code (3 months)

Objectives:

- Build core VS Code extension with **static analysis features**
- Integrate **Jixia static knowledge layer**

Tasks / Deliverables:

- VS Code extension skeleton
- Static theorem metadata hover / tooltip
- Inline display of symbols, dependencies, module graph
- Static autocomplete for tactics / theorem names
- Backend server for static queries and caching
- Unit tests for static query responses

Milestone:

- Developers can view theorem metadata, symbol graph, and dependencies inline in VS Code
-

Phase 2 – On-Demand Step Trace Integration (3–4 months)

Objectives:

- Integrate LeanDojo-style dynamic step trace
- Display step-by-step proof states and subgoal evolution

Tasks / Deliverables:

- Backend LeanDojo hooks / Lean server integration for step traces
- Lazy/on-demand trace request from VS Code extension
- Inline display of proof steps: goal before/after tactic, subgoals
- Cache layer to store traced steps for reuse
- API for retrieving dynamic traces
- Testing on selected theorems / small mathlib files

Milestone:

- Inline step traces are available in VS Code for selected theorems
 - Proof steps cached for performance
-

Phase 3 – LLM / Proof Agent Integration (2–3 months)

Objectives:

- Integrate AI-powered **tactic prediction and proof planning**
- Combine static + dynamic data for LLM retrieval

Tasks / Deliverables:

- Connect LLM/proof agent to backend
- Implement tactic suggestion API for VS Code autocomplete
- Ranked tactic suggestions with confidence scores
- Retrieval-augmented reasoning using static + dynamic data
- Inline / hover display of predicted tactics
- Unit tests and integration tests with step trace

Milestone:

- LLM-powered suggestions available in VS Code
 - Inline tactic recommendations combined with step trace
-

Phase 4 – Optimization & Scalability (2–3 months)

Objectives:

- Improve performance and UX
- Optimize caching and database access
- Prepare for multi-user / cloud deployment

Tasks / Deliverables:

- Performance profiling (VS Code responsiveness, backend latency)
- Optimize on-demand tracing (only trace modified theorems)
- Improve cache DB indexing for faster retrieval
- Support larger Lean projects / team usage
- Optional: cloud deployment with central DB and backend

Milestone:

- IDE extension works smoothly on medium-to-large Lean projects
 - Step trace retrieval and LLM suggestions have low latency
-

Phase 5 – User Testing & Iteration (1–2 months)

Objectives:

- Collect feedback from early users
- Refine UX and AI suggestions
- Prepare for wider release

Tasks / Deliverables:

- Beta release to internal/test users
- User feedback on hover, inline step display, tactic suggestions
- UX improvements and bug fixes
- Documentation for setup and usage

Milestone:

- Product ready for official release / internal deployment
-

Phase 6 – Maintenance & Future Enhancements (Ongoing)

Objectives:

- Keep the tool updated with Lean version changes
- Add new AI features, cloud-based model updates
- Expand dataset for proof agent training

Tasks / Deliverables:

- Continuous integration / deployment setup
- Periodic updates to support Lean4 upgrades
- Integration with additional LLM models or improved retrieval
- Dataset collection for training next-gen proof agents

Milestone:

- Product maintained with ongoing support and feature enhancement
-

Project Timeline Overview (Approximate)

Phase	Duration	Key Deliverables
0 – Preparation	2–3 weeks	Architecture & plan, VS Code extension skeleton
1 – Static MVP	3 months	Jixia static analysis inline in VS Code
2 – Step Trace Integration	3–4 months	LeanDojo step traces displayed inline

Phase	Duration	Key Deliverables
3 – LLM Integration	2–3 months	AI tactic suggestions, retrieval-augmented reasoning
4 – Optimization	2–3 months	Performance tuning, caching, large project support
5 – User Testing	1–2 months	Beta release, UX improvements, documentation
6 – Maintenance	Ongoing	Version updates, feature enhancements, dataset collection

If you want, I can **combine this development plan with the previous project plan and architecture diagram** into a single one-page “Project Plan & Timeline” document ready for executive review.

Do you want me to do that next?



How to make profit from this project

Absolutely — let’s break down **how a hybrid Jixia + LeanDojo IDE project could generate revenue**. There are multiple avenues, ranging from direct monetization to strategic long-term value. I’ll break them into **practical strategies**.

1 Direct Monetization – Product Licensing / SaaS

Options:

1. Paid VS Code Extension / Pro Version

- Free basic version (static analysis via Jixia)
- Paid premium version:
 - On-demand LeanDojo step traces
 - LLM-powered tactic suggestions
 - Advanced caching / cloud storage

2. SaaS Backend Subscription

- Host backend server in cloud
- Users pay per-user or per-project for:
 - Step trace generation
 - LLM-powered proof planning
 - Team collaboration features

3. Enterprise Licenses

- Offer bulk licensing for:
 - Universities
 - Research labs
 - AI / formal verification teams
- Includes:
 - Installation support
 - Private server deployment
 - Custom integrations

Revenue Model: Subscription, per-seat licensing, or one-time license fee

2 Data Monetization

The hybrid system generates high-quality datasets:

- Step-by-step proof traces (dynamic data)
- Symbol / theorem embeddings (static data)
- LLM training datasets (retrieval + imitation learning)

How to monetize:

1. Sell curated datasets to:
 - AI companies
 - Academic research groups
 - Open-source LLM projects targeting theorem proving

2. Use datasets internally to train a proprietary LLM

- Then offer LLM-as-a-service for:
 - Automated theorem proving
 - Tactic recommendation
 - Proof assistance

Revenue Model: Licensing datasets or API access

3 Consulting & Custom Solutions

- Offer integration and consulting services for:
 - Universities (formal methods courses)
 - Companies needing formal verification
 - AI research groups integrating Lean workflows
- Could include:
 - Custom LLM model tuning
 - On-premise backend deployment
 - Training programs for developers

Revenue Model: Hourly consulting fees or project contracts

4 Educational / Academic Licensing

- Target educational institutions:
 - Lean courses
 - Graduate-level theorem proving / formal verification
- Provide:
 - Full IDE with inline hints
 - Step trace visualization for teaching
- Offer tiered pricing:
 - Free for students
 - Paid for department / lab licenses

Revenue Model: Volume-based educational licenses

5 Partnerships with AI Companies / Research Labs

- Collaborate with:
 - LLM developers (OpenAI, Anthropic, Hugging Face)
 - Formal verification startups
- Provide:
 - Step-level datasets
 - Integrated IDE + proof agent
 - Access to unique proof telemetry

Revenue Model: Revenue share, licensing, co-development contracts

6 Freemium + Upsell Strategy

- Free version:
 - Only static analysis, basic theorem info
- Paid upgrades:
 - Step trace capture
 - AI-assisted tactics
 - Cloud-based collaboration
 - Large repository support

Advantage: Attract wide user base first → convert to paying users

7 Long-Term Strategic Value

Even if immediate revenue is limited, the project can create **strategic assets**:

- Proprietary proof step datasets → valuable for AI research
- First-mover advantage in **Lean + AI IDE market**
- Potential acquisition by:
 - AI startups
 - Formal verification tool vendors
 - Cloud-based code assistant companies

Summary of Profit Channels

Channel	How to Monetize	Examples
Paid VS Code Extension	Pro version, premium features	Step trace + LLM hints
SaaS Backend	Subscription / per-seat	Cloud-based proof server
Data Sales	Datasets, embeddings, LLM training	Step-level proof traces
Consulting	Integration, deployment, training	Universities, labs, companies
Educational Licenses	Volume-based pricing	CS / math departments
Partnerships	Co-development, licensing	AI / formal verification companies
Freemium Upsell	Attract users, then convert	Static free + dynamic premium
Strategic / Acquisition	Long-term IP value	AI research / startup acquisition

Key Takeaways for Maximizing Profit:

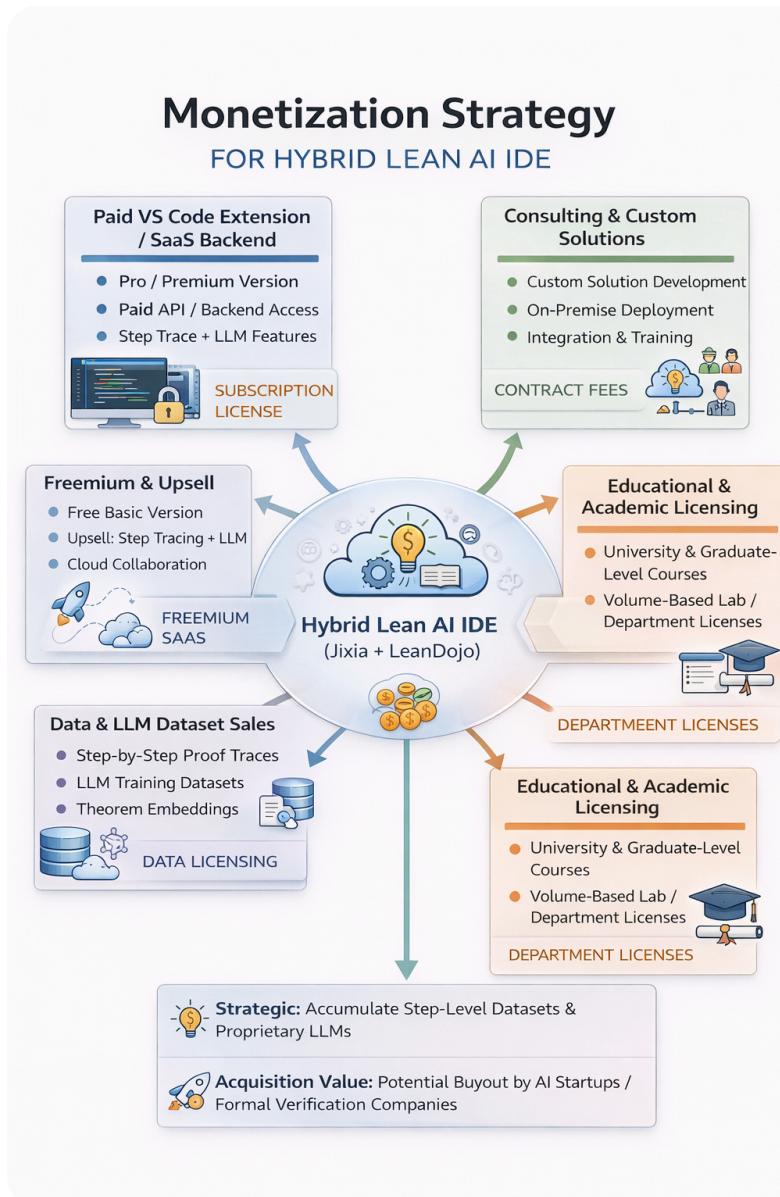
1. Start **freemium** → attract users → upsell advanced features.
2. Collect **step-level proof data** → monetize via datasets or AI training.
3. Offer **SaaS / cloud backend** for scalable subscription revenue.
4. Target **education + enterprise** → easier adoption and higher margins.
5. Maintain **IP in datasets and LLM models** → strategic leverage for partnerships or acquisition.

If you want, I can **create a one-page “Monetization Strategy Diagram”** showing all revenue channels visually, so it's ready for a board review.

Do you want me to do that next?

yes

Image created • Monetization strategy for Hybrid AI IDE



how to select software license for this product?

Choosing a **software license** for your hybrid Jixia + LeanDojo IDE project is a **critical decision**, because it affects **how you can monetize, distribute, and protect IP**. Since your project has both **open-source components (Jixia / LeanDojo)** and **proprietary elements (LLM datasets, SaaS backend)**, the strategy should be carefully planned.

Here's a structured approach:

1 Decide Your Licensing Goals

Ask yourself:

Question	Purpose
Will the source code be open for others?	Community adoption, contributions
Will you sell a commercial version?	Direct revenue from users
Will you host a SaaS backend?	Subscription revenue
Do you want to retain IP on LLM datasets or AI models?	Proprietary advantage
Do you want compatibility with LeanDojo / Lean ecosystem licenses?	Legal compliance

2 Consider License Types

A. Open Source Licenses

- **MIT / BSD / Apache 2.0**
 - Permissive, allows commercial use
 - Good if you want a **freemium extension** for VS Code

- Apache 2.0 protects against patent claims
- **GPL / AGPL**
 - Strong copyleft — derivative works must also be open-source
 - Could conflict with monetization unless you fully control the ecosystem

Best for:

- Static analysis portion (Jixia integration)
 - Non-proprietary UI code
 - Community extensions
-

B. Proprietary / Commercial Licenses

- Custom EULA (End User License Agreement)
 - Restricts redistribution / copying
 - Allows paid features, SaaS, or dataset licensing
- SaaS terms
 - Users access functionality online but cannot extract source code or datasets
 - Supports subscription-based revenue

Best for:

- LLM datasets
 - LeanDojo step-trace engine + cache backend
 - Premium VS Code features
 - AI-assisted tactic recommendations
-

C. Mixed / Dual Licensing

- **Strategy:**
 - Core static analysis and IDE integration → MIT or Apache 2.0 (community adoption)
 - Step trace, LLM integration, and SaaS backend → Proprietary license
 - Allows you to **gain community traction** while retaining **commercial control over premium features**
-

3 Special Considerations

1. LeanDojo / Lean Dependencies

- Check LeanDojo's license (likely MIT/Apache).
- Ensure derivative works comply with upstream license.

2. LLM / Dataset IP

- Proprietary license recommended.
- You can allow access via API / SaaS, not raw download.

3. SaaS vs Local Extension

- If using SaaS backend: you can host proprietary features without distributing code
- If local install: proprietary license or dual license needed to prevent redistribution

4. Open Source Community Benefits

- Open-source the IDE extension, documentation, or basic static analysis
- Increases adoption and credibility

4 Recommended Licensing Strategy

Component	Suggested License	Reason
VS Code extension core (UI, static analysis)	MIT / Apache 2.0	Encourage community use and contributions
Dynamic step trace engine (LeanDojo hooks)	Proprietary	Monetizable feature, retains IP
LLM integration / AI suggestions	Proprietary	Protect datasets & model IP
SaaS backend	Proprietary + TOS	Subscription revenue without code distribution
Datasets / embeddings	Proprietary	Can be sold/licensed separately

Summary: Dual license approach works best — **community-accessible frontend + proprietary premium backend & AI.**

5 Practical Steps

1. Identify all third-party dependencies and check their licenses
 2. Separate your code into **open-source vs proprietary modules**
 3. Draft:
 - MIT / Apache LICENSE for frontend / static layer
 - Proprietary EULA / SaaS TOS for backend / AI services
 4. Clearly document licensing for users, especially for VS Code marketplace
 5. Include **patent and IP clauses** if AI/ML models are included
-

If you want, I can **draft a recommended license map diagram** showing **which component uses MIT, which is proprietary**, so it's visually clear for your legal & management review.

Do you want me to do that?



Do you want compatibility with LeanDojo / Lean ecosystem licenses: yes, Will the source code be open for others: yes but limit. Will you host a SaaS backend: yes

Perfect — thanks for clarifying! Based on your answers:

- **Compatibility with LeanDojo / Lean ecosystem** → Must respect upstream license (likely MIT/Apache).
- **Source code open for others, but limited** → Partial open-source / dual-license strategy.
- **SaaS backend** → Proprietary for monetization and IP protection.

Here's a **recommended licensing strategy** tailored for your project:

1 Component Licensing Map

Component	Suggested License	Notes
VS Code Extension – core UI & static analysis	MIT / Apache 2.0	Permissive; compatible with Lean ecosystem; allows community contributions
Optional premium features in extension (inline step trace, AI hints)	Proprietary / Commercial	Only available to paying users; restrict redistribution
Dynamic Proof Trace Engine (LeanDojo hooks)	Proprietary / Commercial	Monetizable; SaaS backend or paid plugin; retains IP
LLM Integration / AI Agent	Proprietary	Protect model, embeddings, datasets; access via API/SaaS
SaaS Backend / Cloud Services	Proprietary + Terms of Service (TOS)	Users access features online; cannot extract code or datasets
Datasets / Embeddings	Proprietary	Can be licensed separately to other researchers or AI teams

Key Idea:

- Dual-license approach: open-source **safe/core features** for adoption, proprietary **premium features** for revenue.
- Ensures **LeanDojo / Lean ecosystem compatibility** while allowing monetization.

2 Open Source Considerations

- Keep **static analysis + IDE integration** open-source.
- Clearly mark **premium features as proprietary** in code and marketplace description.
- Provide a **limited license** for contributors (e.g., MIT + contribution CLA) to control IP usage.
- Upstream license compliance: If LeanDojo code or APIs are used, include **proper attribution**.

3 SaaS Backend Strategy

- Host **premium functionality** (step trace + AI suggestions) online.
 - Users subscribe for access → prevents redistribution of proprietary logic or datasets.
 - Use a **custom TOS / EULA** for SaaS service.
-

4 Recommended Actions

1. Split your codebase clearly: **Open vs Proprietary**
 2. Add LICENSE files:
 - MIT/Apache for core extension
 - Proprietary license file for premium backend & AI code
 3. Draft **Contributor License Agreement (CLA)** if you accept external contributions
 4. Define **SaaS terms of service** clearly to protect proprietary algorithms & datasets
 5. Include **IP notice for LLM datasets** to prevent unauthorized copying
-

💡 **Visual Idea:** We could make a **license map diagram** showing which part of the product is open-source, which is proprietary, and how SaaS fits in, which is perfect for legal & management review.

Do you want me to make that diagram next?



how Clearly mark premium features as proprietary

Marking **premium features as proprietary** clearly is essential to avoid confusion, ensure legal compliance, and signal to users what's free vs paid. Here's a **structured approach**:

1 Separate Code Structure

Folder / Module separation:

 Copy code

```
project-root/
├── src/
│   ├── core/           # Open-source static analysis & basic UI
│   ├── premium/        # Proprietary features (step trace, AI hints)
│   └── shared/         # Shared utilities
├── LICENSE_OPEN.md    # MIT / Apache License for core
└── LICENSE_PROPRIETARY.md # Proprietary License for premium
└── README.md
```

Key idea: Keep proprietary features **physically separate** in the codebase. Avoid mixing them with open-source core.

2 Add License Headers in Code

For **premium modules**, include a clear header in each file:

```
python  Copy code

# =====
# Proprietary Module - Premium Feature
# Part of Hybrid Lean AI IDE
# © 2026 Your Company / Project Name
# Unauthorized copying, redistribution, or use is prohibited.
# License: Proprietary
# =====
```

For open-source modules:

```
python  Copy code

# MIT License
# © 2026 Your Company / Project Name
```

Effect: Anyone reading the code immediately knows which files are proprietary.

3 VS Code Marketplace & Documentation

- Clearly label features in UI and marketplace description:

"Core static analysis features are free and open-source. Premium features (step-level proof traces, AI tactic suggestions) require a paid subscription or license."

- Include a **feature comparison table**:

Feature	Free	Premium
Static analysis & symbol graph	✓	✓
Step-level proof trace	✗	✓
AI-assisted tactic suggestion	✗	✓
Cloud caching / SaaS backend	✗	✓

4 Build-Time / Access Control

- Use **feature flags** or **modular packaging**:
 - Core open-source modules can run without premium features
 - Premium modules require license check or SaaS login
- Example:

typescript

 Copy code

```
if (user.hasPremiumAccess()) {  
    loadPremiumModule();  
}
```

- Ensures premium code isn't accidentally accessed in free installations.

5 Separate Distribution / Packaging

- Publish **two packages** if possible:
 1. Open-source package (core VS Code extension)
 2. Paid/proprietary plugin or SaaS backend (premium features)
- Avoid including proprietary code in the open-source package.

6 Contributor Guidelines

- If accepting contributions, clarify in **CONTRIBUTING.md**:
 - Contributions to core open-source modules are welcome
 - Premium modules are proprietary; external contributions are not accepted
-

7 Optional: Obfuscation / API Access

- For SaaS backend features, you can **hide premium logic behind API calls**:
 - Users access premium feature remotely
 - Code is never shipped locally
 - Strong protection for IP / AI models
-

Summary of Steps

1. **Folder/module separation**: keep proprietary features separate
 2. **License headers**: mark premium files clearly
 3. **Documentation & Marketplace**: explain free vs premium
 4. **Feature flags / modular loading**: enforce access control
 5. **Separate packaging / distribution**: avoid shipping proprietary code in open-source
 6. **Contributor guidelines**: restrict external contributions to premium features
 7. **Optional SaaS / API**: protect sensitive algorithms and datasets
-

If you want, I can **draw a visual diagram showing the hybrid codebase with open-source vs proprietary layers and SaaS backend**, which is perfect for legal and product review.

Do you want me to do that?



yes

Image created • Hybrid codebase diagram with licenses

