Track IT — Case Study

Portfolio:https://www.arjunportfolio.xyz **Product:** https://track-it-land.vercel.app/

SECTION 1 — Problem Statement

As AI and machine learning workflows accelerate across industries, data scientists increasingly rely on rapid, iterative experimentation within notebook environments such as Jupyter, Google Colab, and VS Code. These environments are highly flexible, but they lack one critical capability: a reliable, automatic way to trace and reproduce data preprocessing steps.

For notebook-heavy teams, the preprocessing stage is the most fluid and least documented part of the machine learning lifecycle. Unlike modeling frameworks—where tools like MLflow, Weights & Biases, and SageMaker provide structured tracking—there is no lightweight solution that captures how raw data evolves through transformations, feature engineering, and exploratory manipulations.

As a result, users face three persistent problems:

1. Loss of Experiment Context

Data scientists frequently forget:

- which sequence of transformations produced a particular dataset
- why certain preprocessing decisions were made
- which experiments succeeded or failed

According to my interviews and research signals, 7 out of 10 practitioners struggle to recall or explain the full set of preprocessing experiments they ran, especially across multiple notebook sessions.

2. Reproducibility Breakdowns

When experimentation happens quickly, intermediate states get overwritten and undocumented.
Teams report losing ~2 hours per experiment reconstructing past steps—manually re-running cells, comparing notebook versions, or diffing scripts to rediscover the transformations originally applied.

This leads to:

- inconsistent outputs across runs
- difficulty debugging upstream issues
- accidental reuse of stale or incorrect datasets

3. Collaboration Friction

In small teams, workarounds such as:

- spreadsheets
- manual logs
- screenshots of notebook cells
- ad-hoc script folders

are incomplete, inconsistent, and heavily dependent on individual discipline.
New team members often lack visibility into the lineage of the data they inherit, slowing onboarding and increasing the risk of duplicated work.

Why This Problem Is Growing NOW

The pain is intensifying due to structural industry shifts:

- LLM workflows increase experimentation volume, multiplying preprocessing branches.

- Regulated industries (finance, healthcare) require reproducibility and auditability of ML workflows.
- AI teams are scaling faster, increasing the number of contributors touching datasets.
- Notebook-first development is becoming the norm, yet notebooks remain inherently ephemeral and unversioned at the transformation level.

Why Existing Tools Fall Short

Current tools either:

- track models, not preprocessing steps (MLflow, W&B), or
- capture pipeline-level lineage, not notebook-level experimentation (Airflow, DataHub, Pachyderm), or
- version files, not dataset evolution (Git, LakeFS).

None solve the thin slice where the pain is highest:

> capturing real-time, transformation-level lineage inside exploratory notebook workflows.

The Resulting Impact

Teams experience:

- slowed iteration cycles
- inconsistent modeling results
- wasted engineering hours
- difficulty reproducing experiments for audits
- fragmented institutional knowledge

SECTION 2 — User Research & Problem Understanding

To validate whether the problem extended beyond my personal workflow, I conducted structured qualitative research using a Mom Test–inspired approach. Instead of asking users whether they "wanted a tool," I focused on uncovering their actual behaviors, pains, and current workarounds.

My research spanned both broad community channels and direct interviews with practitioners.

2.1 Research Channels

To capture diverse perspectives, I intentionally sourced insights from multiple communities that represent different parts of the DS/ML ecosystem:

Public Communities (Exploratory Discovery)

- Reddit — r/MLQuestions and r/AskDataScience
- Hacker News (Show HN / Ask HN)
- Slack communities — Data Science & ML channels
- Indie Hackers / Indie Stack

These channels provided:

- high-volume qualitative input
- real-world pain points
- honest, unfiltered feedback

https://www.reddit.com/r/MLQuestions/comments/1odn6qp/data_scientists_ml_engineers_how_do_you_keep/

https://www.reddit.com/r/askdatascience/comments/1odn05i/data_scientists_ml_engineers_how_do_you_keep/

Direct Interviews (Deep Discovery)

I conducted 2 in-depth interviews with practicing data scientists to probe:

- how they structure their experimentation
- how they document preprocessing
- failure modes in reproducibility
- collaboration issues

These interviews validated patterns observed in broader channels.

2.2 What I Asked (Mom Test Alignment)

Following Mom Test principles, I avoided "solution-leading" questions.

Instead, I asked:

- "Walk me through the last time you ran multiple preprocessing experiments."
- "How did you keep track of what you tried?"
- "What made it hard to revisit past work?"
- "What tools did you use? What broke down?"
- "When collaborating, what usually goes wrong?"

These questions uncovered behaviors, not opinions.

2.3 Key Insights (Patterns from 40+ Comments + Interviews)

Across Slack, Reddit, Hacker News, and interviews, consistent themes emerged:

Insight 1 — Users frequently lose track of preprocessing logic

Multiple practitioners echoed this:

"I often lose track of filtering logic, feature engineering, and other preprocessing steps."
(Slack comment)

This validates that the problem is not unique to me.

Insight 2 — Experimentation is fragmented and poorly documented

Common issues included:

- overwriting notebook cells
- re-running experiments without recording differences
- difficulty explaining why certain transformations were made

One user noted that they rely on custom comments inside MLflow just to capture thinking — an indicator that no native solution exists.

Insight 3 — Teams struggle with reproducibility

When scripts evolve or team members touch the same notebook, reproducibility breaks:

- "We use MLflow + Git + DVC, but preprocessing still gets lost."
- "It takes too long to reconstruct how the dataset was built."
- "New team members can't understand the lineage of what happened earlier."

Directionally, I observed:

- Most users reported losing multiple hours per week rediscovering past steps.
- Recreating old preprocessing pipelines is a repeated pain across teams.

Insight 4 — Workarounds exist, but none fully solve the problem

Users mentioned:

- Custom logging functions
- Manually printed logs in notebooks
- Ad-hoc script folders named “final_v3,” “final_final,” etc.
- Spreadsheets documenting experiments
- Using SQL temporary tables to track transformations
- MLflow notes as a pseudo-lineage tool
- YAML-based tracking via Azure MLTable (misaligned with DS workflows)

These solutions were:

- inconsistent
- manual
- fragile
- dependent on personal discipline

The existence of *so many hacks* indicates an unsolved but meaningful problem.

Insight 5 — Existing tools don’t track preprocessing lineage

Users repeatedly said:

- “We use MLflow, but it doesn’t track transformations.”
- “Git doesn’t help with notebook states.”
- “DVC solves data versioning but not the steps taken.”

- “Airflow is too heavy for experimentation.”

This reinforced a clear gap:

> No existing tool captures transformation-level lineage during notebook experimentation.

2.4 Quantitative Signals (Directional but Valuable)

Based on aggregated themes:

- ~70% of practitioners reported difficulty remembering or explaining preprocessing decisions.
- 2+ hours per experiment were commonly lost retracing steps.
- Nearly all users relied on manual documentation or inconsistent workarounds.
- 0 users reported having a unified preprocessing lineage tool.

These signals justified moving toward solution exploration.

2.5 What Users Want (Inferred Needs)

From user pain points, implicit needs emerged:

- “I don’t want to change how I work.” → low-friction tool
- “It should automatically capture what I do.” → passive lineage tracking
- “I want visibility into how my data changed.” → transformation graph
- “I want summaries so I don’t have to reread notebooks.” → LLM-generated summaries
- “Privacy matters.” → local-first implementation

This shaped the direction of the MVP.

2.6 Key Assumptions to Validate Next

Although initial research showed strong qualitative signals of pain, several critical assumptions still needed validation before committing to an MVP.

Assumption 1 — Severity of the Problem

Is preprocessing lineage a burning, high-frequency pain, or merely an annoyance?

Signals so far:

- Users lose time and context
- Workarounds exist
- But teams may tolerate current friction

Hypothesis:
If the problem is severe, users should report frequent reproducibility failures or repeated manual reconstruction of past experiments.

Risk:
If severity is low, adoption and retention will be weak.

Assumption 2 — Willingness to Pay

Even if the problem exists, will users pay for a dedicated tool?

Questions to validate:

- Would individuals pay? Only teams?
- Is this a workflow tool or an MLOps add-on?
- Should it be SaaS or open-source with paid features?

Risk:
If users perceive lineage as “nice to have,” monetization may be limited.

Assumption 3 — Market Gap Confirmation

Existing tools (MLflow, W&B, DVC, DataHub, Airflow, Azure MLTable) partially solve related problems.

Key uncertainty:

- Is there truly a “white space” between preprocessing tracking and full MLOps pipelines?

Hypothesis:
The gap exists at the notebook-focused, transformation-level granularity.

Risk:
If existing tools or upcoming features cover this slice, the market may be smaller than it appears.

Assumption 4 — Simpson’s Paradox (Sample Bias)

Early research came from:

- 2 interviews
- A few Slack discussions
- Reddit + Hacker News posts

Low volume of responses could reflect:

- niche problem
- poor post visibility
- lack of awareness
- people silently tolerating pain
- or genuinely low demand

Risk:
If only a subset of users experiences the pain acutely, the total addressable market may be smaller.

Assumption 5 — Engagement Signals (Views vs Comments)

Your threads received many views but low comment engagement.

This can mean:

Interpretation A — Market is small
Fewer people face this exact problem.

Interpretation B — Market is large but silent
People don't know how to articulate preprocessing lineage pain
or
They've normalized the struggle.

This is VERY common in DS workflows — problems persist for years because no one thinks they're solvable.

What low engagement *does not* mean:
It does not necessarily mean low demand.
Posts on Reddit/HN often need:

- correct timing
- correct title phrasing
- correct community visibility
  to get traction.

So low comments ≠ no problem.
It just means you need deeper, targeted outreach.

SECTION 3 — ICP Definition

Based on qualitative research, community signals, and direct interviews, it became clear that the pain of losing preprocessing lineage is not evenly distributed across all data practitioners.
Some users feel the pain occasionally, while others feel it constantly and painfully.

To build a focused, high-adoption MVP, I defined the Ideal Customer Profile (ICP) by analyzing:

- workflow patterns
- tooling ecosystems
- frequency + severity of pain
- technical maturity
- willingness to adopt lightweight tools
- monetization potential

This resulted in one primary ICP, plus two secondary segments that can be expanded into later.

3.1 Primary ICP — Notebook-First Data Scientists in Small Teams (1–5 People)

This is the segment where the pain is sharpest, most frequent, and least solved.

Characteristics

- Work primarily in Jupyter notebooks, Google Colab, VS Code notebooks
- Run rapid, iterative preprocessing experiments
- Often the only or one of few data practitioners at the company
- Have *no formal MLOps platform*

- Need reproducibility but lack time/resources to build infrastructure
- Balance research, experimentation, and light engineering

Why They Are the Best Early Users

- Preprocessing changes frequently
- Context gets lost every day
- Workarounds like spreadsheets or print logs are common
- Low workflow inertia → willing to try tools
- High ROI for time saved
- Less bureaucratic tools approval

Pain Severity Score: Very High

Willingness to Pay: Moderate

Teams may pay $10–40/user/month if:

- it saves many hours weekly
- it improves collaboration
- it reduces experiment chaos

Good for MVP? YES

- Easy adoption
- Clear value
- Low friction
- Immediate impact

3.2 Secondary ICP #1 — ML Engineers + Applied Scientists in Growing Teams (5–20 People)

These users are more engineering-heavy and often collaborate in:

- shared repos
- shared notebooks
- mixed pipelines (notebooks + Python modules)

Why They Experience the Pain

- Reproducibility gaps slow down onboarding
- Knowledge loss when experiments are not documented
- Difficult to understand how a dataset was transformed before modeling
- Git alone cannot capture experiment-level lineage

Challenges for Early Adoption

- They may already use MLflow, DVC, DataHub, or Airflow
- They may consider preprocessing lineage “important but not urgent”
- Teams sometimes prefer internal tools

Willingness to Pay: High (team budgets exist)

But: requires a more polished, team-friendly product.

Good for MVP? Maybe later

This segment is better suited for:

- Team dashboards
- Collaboration features

- Cloud sync
- Compliance mode

3.3 Secondary ICP #2 — Regulated Industry Teams (Finance, Healthcare, Biotech)

These teams care deeply about:

- reproducibility
- audit trails
- compliance
- traceability
- documentation

Pain Severity: Extremely High

They often face:

- fines
- audit failures
- broken ML governance

Why They Are Not MVP Users

They require:

- enterprise features
- SOC2/HIPAA
- role-based access

- audit logs
- versioning guarantees

This is a future enterprise segment, not a first target.

Willingness to Pay: Very High

Future SaaS expansion opportunity.

SECTION 4 — Market Landscape & Competitive Gap

4.1 Competitor Landscape Overview

The ML tooling market breaks down into five relevant categories:

1. Experiment Tracking Tools (e.g., MLflow, Weights & Biases)
2. Data Versioning Tools (e.g., DVC, LakeFS)
3. Pipeline Orchestration Tools (e.g., Airflow, Kedro, Prefect)
4. Enterprise Data Lineage Platforms (e.g., DataHub, Amundsen, Collibra)
5. Notebook Environments (e.g., Jupyter, Colab, VS Code notebooks)

Below is what each category covers — and critically, where each falls short

Category	Examples	What They Track	What They *Don't* Track	Gap TrackIT Fills
Experiment Tracking	MLflow, W&B	Models, metrics	Preprocessing lineage	Notebook-first lineage
Data Versioning	DVC, LakeFS	Dataset snapshots	Step-by-step transformations	Transformation evolution
Orchestration	Airflow, Prefect	Pipelines, DAGs	Ad-hoc experiments	Early-stage exploration
Enterprise Lineage	DataHub, Collibra	System flows	Notebook logic	Micro-level lineage

Notebooks	Jupyter, Colab	Code execution	History of transformations	Automatic tracking

4.2 Identified White Space: “Exploration-Phase Lineage”

Across all categories, nobody owns the intermediate layer between:

- rapid notebook experimentation
  and
- production-ready pipelines

This is where most data science work happens — and where most reproducibility failures occur.

This white space is characterized by:

- dynamic, evolving transformations
- partial execution
- branching experiments
- fragmented documentation
- collaboration difficulties

TrackIT is purpose-built for this layer.

4.3 Why Now? (Market Timing)

Several structural trends make this opportunity timely:

1. Explosion of LLM experimentation

Data scientists run *far more* preprocessing variations now than 5 years ago.

2. Notebook-first workflows dominate AI development

Tools like Jupyter, Colab, Kaggle, and VS Code notebooks are the default environment.

3. Auditability and reproducibility are becoming critical

Especially in:

- finance
- healthcare
- enterprise AI deployments

4. MLOps is maturing, but preprocessing lineage remains unsolved

Vendors focus on models and data governance, not exploratory workflows.

5. AI agents require structured lineage to reason about pipelines

Future agents need the historical context of transformations.

This timing makes TrackIT both relevant and strategically differentiated.

4.4 Competitive Risk Assessment

Risk 1 — MLflow or W&B could add preprocessing lineage

Likelihood: moderate
Mitigation: own the notebook-first segment before they move upstream.

Risk 2 — Users may rely on manual logging + Git

Likelihood: high
Mitigation: TrackIT must deliver *10x better* convenience.

Risk 3 — Market may appear small

But history shows:

- Git started small (developer niche)

- Streamlit started small (DS tool)
- HuggingFace started small (NLP toolkit)

DS tooling often scales from niche → mainstream.

SECTION 5 — Key Insights & Problem Themes

Following user interviews, community discussions, and hands-on experimentation, several clear themes emerged that explain why preprocessing lineage consistently breaks down in notebook workflows. These insights reveal not only the existence of the problem, but *why* it has persisted despite the growth of MLOps tooling.

Insight 1 — Preprocessing Is the Least Documented and Most Fragile Stage of the ML Lifecycle

Across every feedback channel, users consistently reported:

- difficulty remembering transformations
- inconsistent documentation practices
- lack of reliable history for exploratory steps

While modeling stages are tracked with MLflow, W&B, or experiment logs, preprocessing lives in an ephemeral zone between ideation and production. This stage changes constantly — and notebooks offer no native mechanisms to capture that evolution.

This insight validated that preprocessing lineage is a blind spot in current DS tooling.

Insight 2 — Notebooks Accelerate Experimentation but Erase History

Users love notebooks because they allow:

- rapid iteration
- branching experiments
- inline visualizations
- exploratory workflows

But these same strengths create systemic weaknesses:

- re-running cells overwrites historical states
- order of execution becomes non-linear
- hidden states accumulate
- partial execution leads to confusion

Multiple users said:

> “I often lose track of filtering logic, feature engineering, and other preprocessing steps.”

This is a structural flaw in notebook architecture — not a user mistake.

Insight 3 — Reproducibility Failures Consume Meaningful Engineering Time

Across research activities, a pattern emerged:

- Users repeatedly rebuild preprocessing steps
- Teams spend hours retracing decision-making
- Collaboration amplifies confusion
- Debugging upstream issues becomes slow and unreliable

Directional signals:

- Users reported ~2 hours lost per experiment reconstructing past transformations
- 70%+ of responses indicated difficulty explaining or reproducing preprocessing logic

This is not a one-off inconvenience — it is a recurring tax on productivity.

Insight 4 — Workarounds Exist, but They Are Manual, Inconsistent, and Fragile

Users employ a patchwork of solutions:

- commented logs inside functions
- manual notes in MLflow
- screenshots of notebook cells
- ad-hoc scripts named “final_v4_fixed2”
- spreadsheet trackers
- SQL temporal tables (rare but mentioned)
- YAML-based MLTable configs (enterprise-only)

The variety of hacks reveals something important:

Everyone solves the problem differently, and no solution is robust.
If users had a reliable automated tool, they would abandon these hacks immediately.

Insight 5 — Existing MLOps and Data Tools Do Not Address This Niche

Every category of tooling misses this exact problem:

- MLflow: tracks models, not preprocessing
- DVC/LakeFS: version snapshots, not transformations
- Airflow/Prefect/Kedro: assume structured pipelines, not exploration
- DataHub/Collibra: enterprise lineage, not notebooks
- Git: versions code, not data evolution
- Jupyter: execution environment only

This positions preprocessing lineage as a white-space opportunity — a layer of the ML workflow not owned by any vendor.

Insight 6 — Pain Is High, but Awareness Is Low

Low-comment engagement on threads could suggest weak demand — but contextual analysis tells a different story:

- Many users think “this is just the way notebooks are.”
- Pain is normalized
- Users cannot articulate a solution because none exists
- Silent majority vs vocal minority effect (common in DS tools)
- High view count vs low comment count is characteristic of latent demand

This phenomenon is common in the success stories of:

- Streamlit
- PostHog
- HuggingFace
- Git (in its very early days)

Users tolerate pain for years — until a simple tool emerges.

Insight 7 — Users Want Zero Workflow Change

Thread patterns showed strong resistance to:

- heavy tools
- adding decorators everywhere
- switching notebook environments
- maintaining YAML pipelines

- configuration-heavy tools

But users responded positively to:

- “automatic”
- “local-first”
- “no setup”
- “passive tracking”

This defined the solution direction clearly:

> The tool must require no behavioral change.

Insight 8 — Collaboration Exposes the Pain More Than Individual Work

In team contexts:

- preprocessing confusion slows onboarding
- handoff friction increases
- shared notebooks create hidden inconsistencies
- lineage gaps lead to contradictory results

Team users felt the problem most intensely, suggesting:

Retention and monetization will come from teams.
MVP adoption will come from individuals.

This insight directly informs the roadmap.

SECTION 6 — Value Hypothesis

Based on user insights, ecosystem gaps, and analysis of existing workarounds, I formulated a set of value hypotheses describing the expected impact of an automated preprocessing lineage tool. These hypotheses guided which solutions to explore, how to prioritize features, and how to evaluate success in the MVP.

These hypotheses are intentionally testable, measurable, and aligned with user behavior uncovered in earlier research.

Value Hypothesis 1 — Automated Lineage Will Significantly Reduce Time Lost Reproducing Experiments

Users reported losing ~2 hours per experiment reconstructing preprocessing steps. Teams repeatedly described:

- re-running notebooks
- comparing old versions
- manually reverse-engineering transformations
- debugging upstream inconsistencies

Hypothesis:

If preprocessing lineage is automatically captured in real time, users will reclaim 1–2 hours per experiment, leading to faster iteration cycles and fewer blocked workflows.

How to measure:

- Time-to-reproduction (before vs after)
- Number of repeated transformations
- Self-reported time saved

Value Hypothesis 2 — Clear Transformation History Will Improve Reproducibility and Reduce Errors

A major theme across interviews and community threads was the difficulty of achieving reproducible results when:

- execution order changes
- hidden states accumulate
- datasets evolve over time
- multiple team members modify the same notebook

Hypothesis:

If TrackIT provides a step-by-step record of how data is transformed, reproducibility failures will decrease, and teams will spend less time debugging.

How to measure:

- Number of reproducibility-related incidents
- Reduction in conflicting results
- Qualitative user confidence in their workflow

Value Hypothesis 3 — Low-Friction, Zero-Workflow-Change Tools Will Drive High Adoption

Users consistently resisted:

- decorators
- manual logging
- custom pipelines
- switching notebook environments

But strongly responded to:

- automatic
- passive
- runs-in-the-background
- local-first

Hypothesis:

If TrackIT requires *zero changes* to user workflows, adoption will be significantly higher vs tools that require configuration or code changes.

How to measure:

- Installation-to-active-use conversion rate
- Retention after 7 / 14 / 30 days
- Number of notebooks tracked per user

Value Hypothesis 4 — LLM-Generated Summaries Will Reduce Cognitive Load and Improve Knowledge Transfer

Users expressed difficulty explaining:

- what they tried
- why certain preprocessing choices were made
- which paths failed
- how datasets were created

Hypothesis:

If TrackIT uses LLMs to summarize preprocessing history, users will spend less time documenting and more time experimenting, and teams will onboard faster.

How to measure:

- Time spent writing documentation (self-reported)
- Onboarding time for new team members
- User satisfaction with summaries

Value Hypothesis 5 — Individual Users Will Adopt the Tool, but Teams Will Derive the Most Long-Term Value

Research showed:

- Individuals feel the pain daily
- Teams feel the pain collectively (onboarding, handoffs, debugging)
- Existing tools serve teams but ignore notebook workflows

Hypothesis:

If TrackIT successfully solves the individual productivity pain, team adoption and monetization will follow as collaboration value becomes apparent.

How to measure:

- Number of users per organization
- Conversion from individual to team accounts
- Requests for collaboration features

Value Hypothesis 6 — Automating Documentation Will Increase Experiment Velocity

Current documentation practices (if they exist) are:

- inconsistent
- manual

- incomplete
- time-consuming

Hypothesis:

If documentation is generated passively, users ship experiments faster and revisit work with less friction.

How to measure:

- Experiment cycle time
- Frequency of revisiting old notebooks
- Summary usage metrics

SECTION 7 — Solution Exploration

To structure this exploration, I evaluated each option using four PM criteria:

1. User Friction — Does this require users to change behavior?
2. Value Delivery — Does it reliably capture preprocessing lineage?
3. Technical Feasibility — Can this be built by one person in weeks, not months?
4. Scalability Potential — Can it evolve into a long-term product?

This produced four viable archetypes of solutions.

Option A — Lightweight Python Library (Decorators / Wrappers)

Description

A Python package where users wrap their preprocessing functions with decorators that automatically log transformations.

Pros

- Easiest to build
- Fastest prototype
- Low engineering complexity
- Open-source-friendly

Cons

- Requires major user behavior change
- Users forget to wrap functions → incomplete lineage
- Does not capture notebook cell-level experimentation

- Breaks down during ad-hoc exploration

Assessment

Low value + high friction → NOT viable for MVP

Option B — Custom Notebook Frontend with Real-Time LLM Summaries

Description

A full notebook environment (like Jupyter) with lineage tracking + embedded LLM summaries.

Pros

- Beautiful UX
- Strong differentiation
- Full control of environment

Cons

- Very high engineering cost
- Requires replicating Jupyter functionality
- Forces users to switch environments (deal breaker)
- Hard enterprise distribution

Assessment

High risk + high effort + low adoption → Not suitable for MVP (maybe long-term)

Option C — Local Background Agent (Notebook Monitoring Service)

Description

A lightweight local service (via Docker or Python agent) that monitors notebook kernel activity, captures cell execution events, logs preprocessing steps, and generates summaries — all without requiring workflow changes.

Pros

- Zero workflow friction (critical)
- Works with existing notebook environments
- Full lineage capture (cell-level + transformation-level)
- Private by default (local-only)
- Strong alignment with user needs
- Reasonable to build in 4–8 weeks

Cons

- Requires the user to start a local service
- Must handle large data safely
- Requires careful kernel event monitoring logic

Assessment

Best balance of value, feasibility, and adoption
Aligns perfectly with notebook-first workflows
Strongest candidate for MVP

Chosen MVP Direction

Option D — Browser Extension for Jupyter/Colab UI Monitoring

Description

A browser extension that intercepts notebook events and logs cell executions.

Pros

- Lightweight
- Easy distribution
- Works across cloud notebook providers (Colab, Kaggle)

Cons

- Brittle and hard to maintain
- No access to kernel-level execution or data states
- Limited ability to capture deep transformations
- Always playing catch-up with UI changes
- Not secure for enterprise notebook environments

Assessment

Low technical feasibility + shallow lineage → Rejected

Option	User Friction	Value Delivery	Feasibility	Scale	Verdict
A: Python Decorator Library	High	Low	High	Medium	Reject
B: Custom Notebook IDE	High	Medium	Very Low	High	Reject
C: Local Background Agent	Low	High	Medium	High	Chosen MVP
D: Browser Extension	Medium	Low	Low	Low	Reject

SECTION 8 — Final MVP Scope

The MVP for TrackIT focuses on validating the core value hypothesis:

> “Automatic lineage tracking and LLM-generated summaries reduce cognitive load and help data practitioners reconstruct experiments faster — without requiring workflow changes.”

To validate this hypothesis quickly, reliably, and with minimal user friction, the MVP includes only the essential components needed to demonstrate end-to-end value.

8.1 MVP Scope (What the MVP Includes)

The MVP consists of four cohesive capabilities:

1. Notebook Discovery UI (Next.js)

A lightweight UI that:

- scans local file system (via Docker volume)
- displays available .ipynb notebooks
- lets the user select a notebook to track

PM Reasoning:
This removes friction and makes the tool accessible to non-technical users.

Applied ML / Engineering Reasoning:
Mapping notebooks via Docker ensures privacy, portability, and simple integration.

---

2. Passive Notebook Tracking

When a user selects a notebook, the backend:

- starts a kernel-watching process
- monitors notebook saves
- extracts cell code, outputs, execution order, timestamps

- computes digests to avoid duplicate entries
- stores lineage snapshots locally

PM Reasoning:
Zero workflow change is a decisive adoption factor.

Applied ML Reasoning:
Accurate lineage metadata is the foundation for all downstream LLM, RAG, and evaluation features.

---

3. Local Log Storage (Privacy-First Architecture)

All logs are stored locally in the backend container:

- JSONL or TXT format
- organized by notebook name
- each snapshot appended incrementally

PM Reasoning:
Makes enterprise adoption possible later.
Lower friction → higher trust → higher activation.

Applied ML Reasoning:
Local storage enables deterministic offline experimentation, LLM chunking, and vectorization later.

---

4. LLM-Generated Summaries via Bedrock

After a session, a user can generate a summary:

- logs fed into structured summary prompt
- Bedrock Llama-3.70B generates:

    - preprocessing steps
    - data transformations
    - experiment rationale
    - overall narrative of the notebook
- summary returned to UI

PM Reasoning:
This validates whether users gain insight + save time.

Applied ML Reasoning:
LLM prompts + chunking unlock future Q&A, RAG, and eval pipelines.

8.2 What the MVP *Explicitly Excludes* (Non-goals)

The following features are intentionally excluded from MVP:

- chat interface / Q&A
- RAG pipelines
- multi-LLM router
- report generation / automated PPT builder
- guardrails or safety layers
- vector database storage
- collaboration or team sync
- dataset version diffs
- cloud sync or user accounts

These belong in V2+, not MVP.

8.3 Why This MVP Scope Is Optimal

The chosen MVP:

- validates the *value hypothesis*
- requires minimal engineering complexity
- demonstrates end-to-end value
- builds infrastructure for future applied ML features
- minimizes adoption friction
- proves desirability before scalability

SECTION 9 — Feature Prioritization

To prioritize features, I used a hybrid framework:

- Must / Should / Could / Won't (MoSCoW — for clarity)
- Value vs Effort (for engineering intuition)
- Hypothesis coverage (for PM rigor)

9.1 Must-Have (MVP Critical)

These features deliver core value + validate hypotheses.

Feature	Reason
Notebook picker	Needed for usability + discovery
Passive tracking	Core of the product; validates lineage hypothesis
Log storage	Needed for reproducibility + summary generation
Summary generation	Tests if LLM value hypothesis works
Basic UI	Needed to close the feedback loop

9.2 Should-Have (High Value, Low Risk, Post-MVP)

Feature	Reason
Log chunking	Summary accuracy improvement
Multiple summaries (per section, per experiment)	Higher granularity
Simple filtering / metadata grouping	Users want to navigate large logs
Export summary (txt/pdf)	Enables sharing + reporting

9.3 Could-Have (Experimental / Applied ML Features)

These are strong differentiators but not essential for MVP validation.

Feature	Reason
Vector DB for logs	Enables RAG + retrieval
Q&A on notebook	Unlocks “AI Experiment Assistant”
Guardrails	Needed for enterprise later

Model adapters (OpenAI, Llama local)	Choice = value but not core MVP
Experiment reports (auto PPT builder)	Great for DS teams + PM reporting

9.4 Won't-Have (Not now / too large)

Feature	Reason
Multi-user collaboration	Needs cloud infra
Team dashboards	Needs backend + auth
Dataset versioning	Whole other product domain
Realtime co-edit lineage	Hard engineering + low MVP ROI

SECTION 11 — Roadmap

V1: Lineage + Summaries(MVP)

Goals:

- Validate automatic tracking works
- Validate summaries deliver value
- Test adoption friction
- Prove problem exists with real users

Features:

- Notebook picker
- Passive kernel tracker
- Local logs
- Bedrock summary generation
- UI
- Basic log filters (optional but easy)

V2: Multi-Model & Evaluation Layer

Goals:

- Improve summary quality
- Increase trustworthiness

- Introduce model choice

Features:

- Add OpenAI + Anthropic API adapters
- Add support for local Llama 3.1 inference
- Add log chunking + stitching
- Add simple evaluation (BLEU/ROUGE or LLM-as-judge)
- Add metadata filtering (date, cell count, tags)

Why it matters:

- Shows *Applied ML engineering depth*
- Shows *PM understanding of model quality*
- Shows *platform extensibility*

V3: RAG + Notebook Q&A Assistant

Goals:

- Transform TrackIT from tool → assistant
- Allow retrieval-based understanding

Features:

- Vector DB for logs
- Structured embeddings of notebook sections
- Retrieval-augmented generation (RAG)

- Q&A interface (“What changed between runs?” “Why did accuracy drop?”)
- Guardrails for prompting

Why it matters:

- Demonstrates *AI Agent* capabilities
- Distinguishes your product from MLflow/W&B
- Shows strong R&D + PM alignment

V4: Reporting & Automation Layer

Goals:

- Automate repeated workflows
- Improve business value

Features:

- Auto-generated PPT based on logs
- Experiment comparison reports
- “Run summary weekly report” emails
- Slack webhook integration

V5 — Team Collaboration + Cloud Sync

Goals:

- Monetization

- Multi-user scale
- Enterprise readiness

Features:

- Auth
- Shared workspace
- Cloud log storage
- RBAC
- Version history across users

V6 — Enterprise & Compliance

Goals:

- Sell to regulated industries
- Become an MLOps lineage platform

Features:

- Audit logs
- Compliance dashboards
- Regulatory lineage tracking
- SOC2

SECTION 12 — Metrics & Instrumentation

To measure the MVP's success and validate key hypotheses, I defined a metrics framework across North Star, Primary KPIs, Secondary KPIs, and Guardrails.

North Star Metric

"% of preprocessing steps automatically captured per active session."

Why it matters:

- Directly tied to the core problem: reproducibility
- Aligns with user value: fewer manual notes
- Predicts retention: reliability → trust → habit-building

Primary KPIs (Value & Adoption)

1. Time-to-Reproduce Experiment

Expected Outcome:
Users reconstruct past work faster after using TrackIT.

This metric validates the core hypothesis.

2. Summary Usage Rate

How often users generate summaries per session.

Signal for:

- perceived value
- summary usefulness
- workflow integration

3. Active Sessions per User (7-day retention)

If users return, TrackIT is sticky.
If they don't, the problem solved isn't strong enough.

Secondary KPIs (Quality & Accuracy)

4. Summary Relevance Score (LLM-as-a-judge or manual rating)

Directional measure of:

- summary accuracy
- content completeness
- logical flow

Guardrail KPIs (Performance & Experience)

1. Notebook Slowdown Impact

CPU/memory overhead of tracking agent.
Must remain negligible (<5% impact).

2. Summary Latency

Time between "Generate Summary" click and UI response.
High latency degrades perceived value.

3. Error Rate (tracking failures, log corruption)

Ensures reliability and trust.

SECTION 13 — Learnings & Reflection

This project was more than a technical exercise — it fundamentally changed how I think about building for data practitioners, designing AI-driven tools, and prioritizing product decisions.

Below are the key reflections.

1. Hidden Pain Is Real Pain

Every DS/ML workflow has undocumented chaos between the lines.
Most users *normalize* this pain, so it doesn't appear loudly in forums.

But once you show people a better way, the value becomes obvious.

---

2. Adoption Beats Features

The winning solution wasn't the most complex — it was the one with the fewest workflow changes.

TrackIT works because users don't need to:

- rewrite code
- remember decorators
- switch tools

This was the biggest insight and guided MVP decisions.

---

3. Local-First Was the Right Bet

Users trust tools that respect privacy.
Especially in ML, where datasets may contain:

- PHI
- PII
- financial data
- proprietary experiments

Local-first made TrackIT instantly trustworthy.

---

4. LLM Value Is About Friction Removal, Not Magic

LLMs aren’t valuable because they’re powerful — they're valuable when they:

- reduce cognitive load
- summarize chaos
- compress knowledge

Summaries turned raw lineage into clarity.
This is *real* AI augmentation.

---

5. Architecture Determines the Roadmap

Because the MVP architecture is modular and local-first:

- RAG becomes trivial
- Q&A assistant becomes natural
- Evals become necessary
- Multi-model becomes plug-and-play