Inside AutoML Reproducibility Hub

A Visual Guide to Reproducible Browser-Based Machine Learning

By **Faramarz Kowsari**

Static Mode
Fast • Deterministic
Shareable

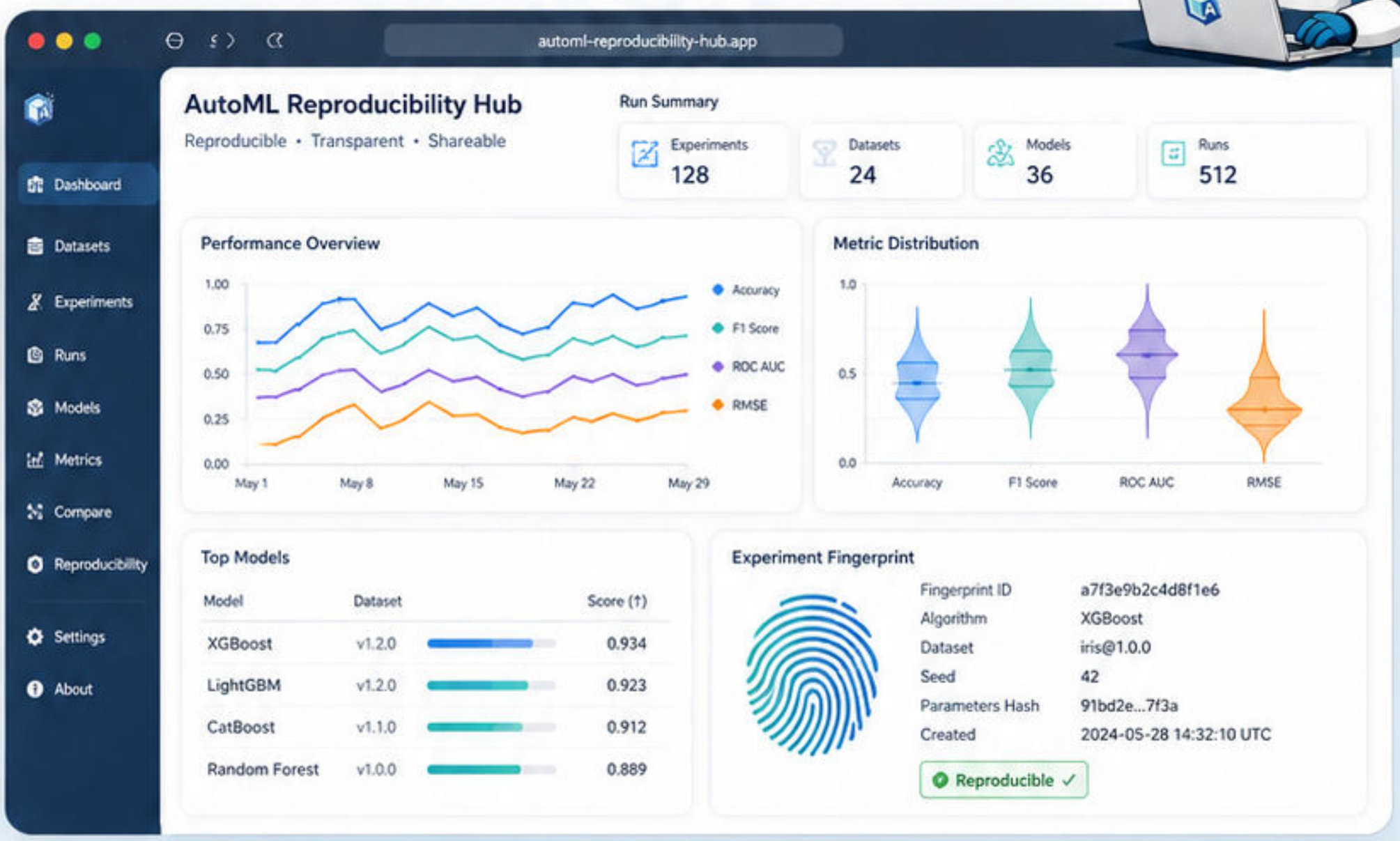
Full Mode
Extended Models
Advanced Features

Built with

TypeScript

Pyodide
(Python in Web)

DuckDB-WASM
(In-Process Analytics)



Seeds

Deterministic Randomness

`seed = 42`

Dataset Versioning

v1.2.0

Immutable & Trackable

Parameters

```
{
  max_depth: 6,
  n_estimators: 200,
  learning_rate: 0.05
}
```

Tracked & Hashed

Metrics

Accuracy
F1 Score
ROC AUC
RMSE

Comprehensive Evaluation

Experiment Fingerprinting

Unique Identity for Every Experiment

Reproducibility

Rerun Anywhere. Same Result.

End-to-End AutoML Pipeline

Load Data → Preprocess → Feature Engineering → Model Selection → Train & Tune → Evaluate → Log & Version → Reproduce & Share

Explore the project on GitHub

github.com/FaramarzKowsari/automl-reproducibility-hub

Inside
AutoML Reproducibility Hub

Project at a Glance

A **browser-first** AutoML laboratory for **reproducible machine learning** experiments.

What is AutoML Reproducibility Hub?

AutoML Reproducibility Hub is an interactive, browser-based platform that helps data scientists run, track, and reproduce AutoML experiments with complete transparency.

It captures every critical detail—data, parameters, environment, and results—so you can verify, compare, and share experiments with confidence.

Browser-Based · Transparent · Shareable · Reproducible

Mini Dashboard Snapshot

automl-reproducibility-hub.app

Run Summary

Experiments	Datasets	Models	Runs
128	24	36	512

Best Model Over Time (by ROC AUC)

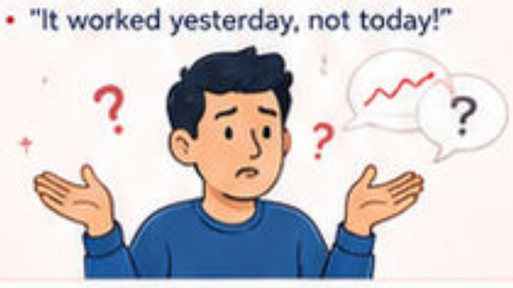
Metric Distribution (ROC AUC)

At a Glance

Run experiments.
Capture everything.
Reproduce anywhere.
Share with confidence.

The Reproducibility Problem in ML

- Results vary across runs
- Hidden changes break comparisons
- Hard to trust or verify models
- Difficult to share and collaborate
- "It worked yesterday, not today!"



Hidden Sources of Inconsistency

- Random seed not fixed
- Dataset version changes
- Different hyperparameters
- Library / version mismatches
- Hardware / environment drift
- Preprocessing variations
- Time & timezone differences

What the Hub Standardizes

- Seed control and tracking
- Dataset versioning & hashing
- Parameter logging & tracking
- Comprehensive metrics capture
- Environment & dependency info
- Fingerprint for unique identity
- Shareable experiment packages

The Core Promise

Same setup.
Same evidence.
Clearer repeatability.

Why Reproducibility Matters

Seed Control
Fix randonmess at the source to ensure runs are repeatable and comparable.
Seed

Dataset Versioning
Track immutable dataset versions and hashes to preserve data lineage.
Dataset Version

Parameter Tracking
Log every parameter and configuration for full experiment transparency.
Parameters

Metric Transparency
Capture all important metrics consistently for honest and apples-to-apples comparison.
Metrics

Environment Awareness
Record libraries, versions, OS, and environment to explain every result.
Fingerprint

Reliable science starts with reproducible evidence.

Without Reproducibility

- Results change, hard to trust
- Lost context & missing details
- Slow debugging & root cause
- Poor collaboration & sharing
- No proof, hard to validate

VS

With AutoML Reproducibility Hub

- Consistent, verifiable results
- Complete context captured
- Fast debugging & root cause
- Seamless sharing & collaboration
- Reproducible anywhere, anytime

Key Concepts Captured

- Seed
- Dataset Version
- Parameters
- Metrics
- Fingerprint
- Shareability

The Reproducible Experiment Lifecycle

1. Prepare Data → 2. Version Dataset → 3. Configure Parameters → 4. Run AutoML → 5. Capture Metrics → 6. Generate Fingerprint → 7. Share & Reproduce

Every run has a unique identity (**Fingerprint**) that proves what was run, where, and how.

Share experiments like a link.
Anyone can reproduce the same result.

Reproducibility turns experiments into trustworthy knowledge.

Trust
Build confidence in every result.

Efficiency
Save time, reduce rework.

Collaboration
Share, review, and build together.

Compliance
Meet standards, ensure accountability.

Better science.
Better models.
Better decisions.

Inside AutoML Reproducibility Hub

A Visual Guide to Reproducible Browser-Based Machine Learning

Architecture & Technology Stack

100% Browser-Native • Reproducible • Transparent • Shareable

Presentation Layer
Interactive UI
Built with React

React UI (Vite + React + Tailwind CSS)
Responsive dashboards, charts, tables, forms, and visualizations.

Application Logic
Typed, Predictable
Maintainable

TypeScript Application Logic
Strongly typed state, services, validation, and domain logic.

```
interface RunSummary {
  model: string;
  dataset: string;
  metrics: Metrics;
  createdAt: string;
}
```

Local State & Storage
Persist Locally
Offline-First

Local State & Persistence
UI state in-memory + long-term persistence via localStorage (immutable snapshots).

```
localStorage
{
  "runs": [...],
  "models": [...],
  "datasets": [...]
}
```

Python Execution
Runs in Browser
No Server

Pyodide Worker (Python in WebAssembly)
Isolated Web Worker runs Python code for ML algorithms, preprocessing, and evaluation.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=200)
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

Data & Analytics Engine
SQL in the Browser
Fast & Analytical

DuckDB-WASM
In-browser analytical database for dataset profiling, transformations, and SQL-style analytics.

```
SELECT target,
       AVG(feature_1) AS mean_f1,
       STDDEV(feature_1) AS std_f1
FROM dataset
GROUP BY target;
```

Browser File Handling
Drag, Drop, Import
Open, Save

Browser File Handling
Upload, parse, and stream CSV/JSON files. Use File System Access API for open/save/export.

CSV data.csv · JSON runs.json · config.json · Drag & Drop or Browse

Computation & Insights
Reusable Metrics
Consistent Results

Metrics Engine
Unified computation of Accuracy, F1, ROC AUC, RMSE, and more with consistent interfaces.

Accuracy	F1 Score	ROC AUC	RMSE
0.934	0.923		0.889

Reproducibility Layer
Capture Everything
Recreate Anywhere

Experiment Manifest Generator
Captures code, params, data schema, metrics, environment, and lineage in a portable manifest.

manifest.json

- Complete experiment fingerprint
- Deterministic re-execution
- Shareable & versionable

Export Layer
Share & Archive
Anywhere

Export Layer
Export runs, manifests, metrics, and datasets to JSON/CSV for sharing or archival.

JSON runs.json · CSV metrics.csv · ZIP bundle.zip · Ready to Share or Archive

Deployment
Static, Fast, Global
No Backend

GitHub Pages Deployment
Fully static site hosted on GitHub Pages. Global CDN, versioned, and always available.

GitHub Repo (main) → GitHub Pages (Build & Deploy) → Live Site (Global CDN)

Two Operating Modes

Static Reference Mode
Fast • Deterministic • Shareable

- Pre-computed demo results
- Zero code execution
- Instant loads, perfect for sharing
- Best for documentation & demos

Full Browser Execution Mode
Extended Models • Advanced Features

- Runs Python via Pyodide in-browser
- Full pipelines & custom datasets
- Reproducible on any modern browser
- Best for exploration & learning

Data & Execution Isolation

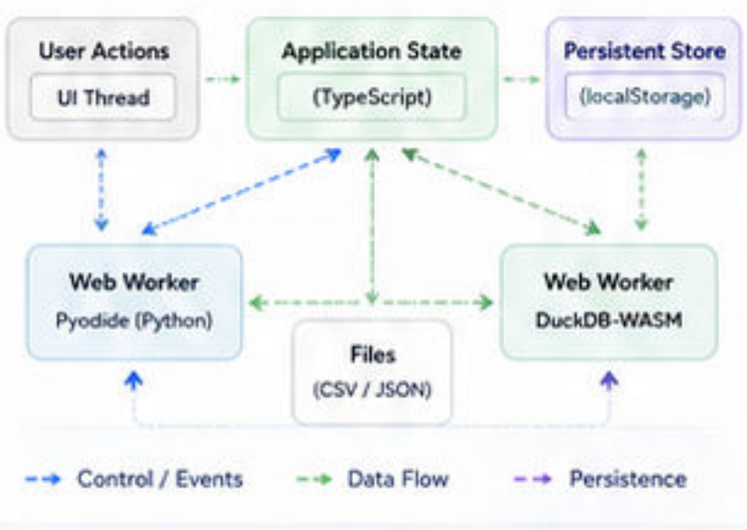
Everything stays on your device.
No data leaves your browser.

Key Data Artifacts

TypeScript	Application code & logic
Pyodide (Python)	ML algorithms & execution
DuckDB-WASM	SQL analytics & profiling
localStorage	Persistent runs & settings
JSON Exports	Manifests, runs, configs
CSV Datasets	Tabular data & results
Web Workers	Isolated, parallel execution

System Flow (High Level)

Architecture at a Glance (Data & Execution Paths)



Why Browser-First?!

No project-owned backend
Everything runs in the browser. Easier to maintain, deploy, and scale.

No hidden server execution
Transparent, inspectable, and trustworthy results.

Portable demos
Share a link, anyone can explore immediately.

Reproducible education
Same environment, same results—every time.

Low deployment friction
Static site on GitHub Pages. Fast, secure, global.

Explore the project on GitHub
github.com/FaramarzKowsari/automl-reproducibility-hub

Static Mode (S)

Fast • Deterministic • Shareable

Static Reference Mode delivers curated, precomputed experiment examples instantly — without running a new model in this browser session.

Important: Reference result — no model was executed in this browser session.
You are viewing static, precomputed results for reference and learning.

What is Static Mode?

Static Mode lets you explore reproducible results that were generated previously and stored in the repository. It loads instantly, uses minimal resources, and provides a reliable reference for comparison, learning, and communication.

Instant Startup
Loads in milliseconds. No pipelines, no waiting.

No Heavy Packages
No model, dataset, or training libraries loaded.

Ideal for Teaching & Demos
Explain results clearly without compute.

Deterministic Reference Examples
Curated, versioned, and fully reproducible.

Metric Previews
See key metrics immediately.

Dataset Cards
Understand data versions at a glance.

Manifest Inspection
View experiment config and lineage.

How Static Mode Works

Curated & Precomputed Experiments
Generated offline and stored.

Stored with Manifest
Config, metrics, artifacts saved.

Loaded Instantly
Static files fetched from repository.

Render Reference UI
Explore metrics, results, and configs.

No Execution
No model run in this session.

Startup Timeline (Typical)

0ms	~50–150ms	~150–300ms	~300–600ms	< 1s
Click Static Mode	Fetch manifest & reference files	Parse & prepare UI data	Render charts, cards & tables	Ready to explore reference results

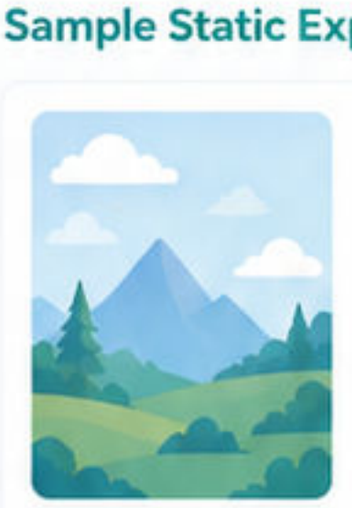
Static Mode vs Full Mode

Static Mode (Reference)	Full Mode (Execution)
Instant load (milliseconds)	Slower startup (seconds–minutes)
No packages loaded	Loads ML/DL packages
No GPU/CPU intensive work	High compute usage possible
Uses precomputed reference data	Runs new model in this session
Deterministic results	Results may vary with changes
Easy to share and embed	Not shareable until completed
Best for learning, demos, docs	Best for research, new experiments

Best Use Cases

- **Classroom Explanation** — Teach concepts using stable, known results.
- **Repository Preview** — Preview what's inside without any setup.
- **Quick Inspection** — Check metrics, configs, and datasets fast.
- **Shareable Examples** — Embed links to exact reference results.

Sample Static Experiment (Reference)



Experiment ID	a73fe9b2cd68f1e6
Seed	42
Dataset Version	v1.2.0
Dataset	XGBoost-Tabular
Model	XGBoost v1.2.0
Created	2024-05-28 14:32:10 UTC
Static Reference	No model executed in this session

Metrics (Preview)
Cross-Validation (5-Fold)

Accuracy	0.934
F1 Score	0.923
ROC AUC	0.912
RMSE	0.889

Average Rank 2.1 / 10
(Lower is better)

Manifest Snapshot

```
{
  "experiment_id": "a73fe9b2cd68f1e6",
  "seed": 42,
  "dataset": "XGBoost-Tabular",
  "dataset_version": "v1.2.0",
  "model": "XGBoost v1.2.0",
  "metrics": ["Accuracy", "F1 Score",
              "ROC AUC", "RMSE"],
  "cv": 5,
  "created": "2024-05-28T14:32:10Z",
  "mode": "static_reference"
}
```

View Full Manifest

Static Mode is perfect when you need speed, stability, and certainty.
Explore, learn, compare, and share — all without running a single model.

Explore the project on GitHub
github.com/FaramarzKowsari/automl-reproducibiity-hub

Full Mode (F)

Full Browser Execution Mode

Full Mode runs the entire machine learning workflow inside your browser. AutoML loads **Pyodide** and a rich Python scientific stack to perform real training, evaluation, and comparison **locally**—no server required.

Your data stays in your browser. **100% private.**

Powered by Browser Python

Pyodide — Python in WebAssembly · NumPy · pandas · scikit-learn · DuckDB-WASM (In-Browser Analytics)

Full Mode Workflow

1. **Load CSV** — Upload or drag your dataset
2. **Profile Data** — Explore columns, types, missing values, stats
3. **Choose Task** — Classification or Regression
4. **Configure Seed & Split** — Set seed, train/valid/test split ratios
5. **Select Model Space** — Pick models, algorithms & search space
6. **Train** — Run training locally in the browser
7. **Evaluate** — Compute metrics, compare performance
8. **Capture Manifest** — Save config, params, env & fingerprint
9. **Export Results** — Download models, metrics, manifests & logs

Browser-Local Execution

```
Loading Pyodide...
Loading NumPy...
Loading pandas...
Loading scikit-learn...
Loading DuckDB-WASM...
Environment Ready!
```

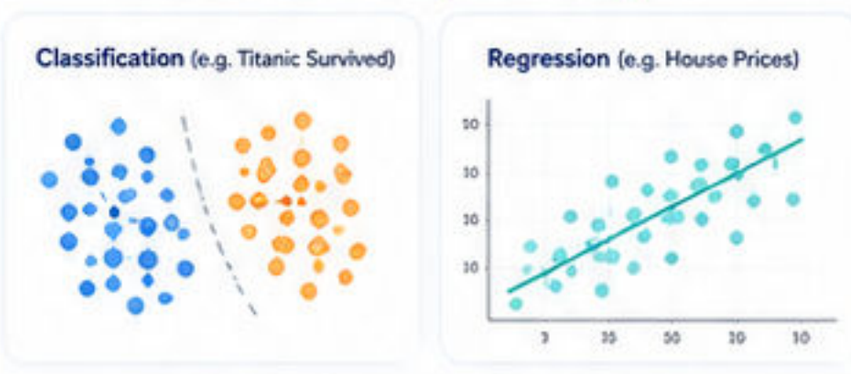
All computation happens locally using WebAssembly.
No data leaves your device.

Private • Secure • Offline-Ready

Real Training Runs

Full Mode performs real model training with cross-validation and hyperparameter search.

Classification & Regression Support



Auto detects task type and evaluates with appropriate metrics.

AutoML Comparison Workflows

Compare many algorithms side-by-side with the same data split, seed and metrics.

Model	Search Type	Best Score (F1)
XGBoost	Bayesian	0.934
LightGBM	Bayesian	0.923
CatBoost	Bayesian	0.912
Random Forest	Randomized	0.889
Logistic Regression	Grid	0.812
KNN	Grid	0.735

Fair Comparison: Same data, same seed, same metrics.

Metric Computation

Comprehensive metrics computed locally.

Classification		Regression	
Accuracy	0.934	RMSE	18.42
F1 Score	0.923	MAE	12.73
Precision	0.921	R^2 Score	0.892
Recall	0.925	MAPE	11.34%
ROC AUC	0.971	MedAE	9.87
Log Loss	0.182	Expl. Var.	0.894

Local Export

Export everything you need, all generated in-browser.

- **Models** (.pkl / .joblib / .json)
- **Metrics** (.json / .csv)
- **Manifest** (.json)
- **Logs** (.txt)

Download All Artifacts

DuckDB-WASM Data Inspection

Fast SQL-powered exploration on your data—fully local.

```
SELECT age, COUNT(*) as cnt
FROM   df
WHERE  age IS NOT NULL
GROUP BY  age
ORDER BY age;
```

age	cnt
20	52
21	63
22	47
23	59
24	61
25	58
...	...

DuckDB WASM — Blazing fast in-browser analytics.

Reproducibility Fingerprint

A unique fingerprint is generated from your experiment to make results verifiable and reusable.

Fingerprint ID	f3e9a7b2c4d8e91f
Algorithm	XGBoost
Dataset	titanic-ml
Seed	42
Parameters Hash	a193f2...c8b1
Created	2024-05-28 14:32:10 UTC

Reproducible

Trade-offs

- **Heavier Startup** — Loading Python + libraries takes longer than Static Mode.
- **More Browser Memory** — Large datasets or many models can use more RAM.
- **Richer Experimentation** — Enables real training, more models, deep comparisons and exports.

Best for deep experimentation & full privacy.

Example: Train a Model in Full Mode

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
df = pd.read_csv('titanic.csv')
X = df.drop('Survived', axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
model = RandomForestClassifier(n_estimators=200, random_state=42)
model.fit(X_train, y_train)
preds = model.predict(X_test)
print('F1 Score:', f1_score(y_test, preds))
```

F1 Score: 0.932

Mode Reminder

Full Mode = Full Power

- Real training
- Rich metrics
- Deep comparisons
- Export everything
- 100% local & private

When to Use Full Mode

- You need real training and hyperparameter search
- You want to compare many models deeply
- Your data is sensitive and must stay local
- You want to export models, metrics & manifests
- You value reproducibility and transparency

Explore the project on GitHub

github.com/FaramarzKowsari/automl-reproducibility-hub

Experiment Lifecycle

End-to-End Reproducibility Pipeline

From raw data to reproducible results — every decision captured, every run repeatable.

1	2	3	4	5	6	7	8	9	10	11	12	14
Ingest Dataset	Detect Schema	Choose Task	Set Seed	Define Split	Configure Preprocessing	Select Model	Set Parameters	Train & Evaluate	Log Metrics	Generate Fingerprint	Save Manifest	Export & Reproduce
data.csv parquet json ...	Rows: 51,234 Cols: 24 Types: int, float, str, bool ...	Classification Regression Forecasting Clustering	seed = 42 np.random.seed(42) random.seed(42) ...	Train: 70% Val: 15% Test: 15% Stratified: ✓	Impute: median Scale: Standard Encode: OneHot ...	XGBoost LightGBM CatBoost Random Forest	n_estimators: 300 learning_rate: 0.05 max_depth: 6 ...	CV: 5-Fold Fit model Evaluate ...	Accuracy: 0.934 F1 Score: 0.927 ROC AUC: 0.972 RMSE: —	fp: a75fe9b2cd dataset: xgboost seed: 42 code: v1.2.0	manifest.yaml config.json schema.json env.lock	Package run Push to Hub Re-run Verify

What Happens at Each Stage

#	Stage	What Is Captured	Why It Matters
1	**Ingest Dataset**	Source path/URL, file checksum, format, row count	Proves the exact data you used
2	**Detect Schema**	Columns, types, nulls, stats, inferred schema	Locks structure & meaning
3	**Choose Task**	Problem type, target(s), metric intent	Clarifies goal & evaluation
4	**Set Seed**	Global seed for all libs (Python, NumPy, etc.)	Controls randomness
5	**Define Split**	Split strategy, ratios, stratification, groups	Ensures identical data partitions
6	**Configure Preprocessing**	Transform steps & fitted artifacts	Recreates identical feature pipeline
7	**Select Model**	Algorithm & library versions	Defines model family & implementation
8	**Set Parameters**	Hyperparameters & search space	Enables exact reconfiguration
9	**Train & Evaluate**	Training config, CV folds, early stopping, scores	Reproduces the training process
10	**Log Metrics**	All metrics, curves, confusion matrix, residuals	Transparent performance evidence
11	**Generate Fingerprint**	Content hash of data, code, params, env	Unique identity for the run
12	**Save Manifest**	Immutable manifest (YAML/JSON) of everything	Single source of reproducibility truth
13	**Export & Reproduce**	Package & publish run, re-run in clean env	Anyone can reproduce the same result

Classification Workflows

Data → Features → Model → Predict

Metrics: Accuracy | F1 Score | ROC AUC | PR AUC | Log Loss

Regression Workflows

Data → Features → Model → Predict

Metrics: RMSE | MAE | R^2 | MAPE | Expl. Var

Leaderboard Comparison

Rank	Model	Dataset	Metric (ROC AUC)	Score
1	XGBoost v1.2.0	CreditRisk	ROC AUC	0.972
2	LightGBM v1.2.0	CreditRisk	ROC AUC	0.965
3	CatBoost v1.1.0	CreditRisk	ROC AUC	0.958

What Makes a Run Reproducible?

- **Data Integrity:** Immutable data + checksums
- **Complete Config:** Every parameter recorded
- **Determinism:** All randomness controlled by seeds
- **Traceability:** Unique fingerprint for every run
- **Environment Lock:** Exact library & system versions
- **Verifiability:** Anyone can re-run & match results

If any one of the above is missing, the run is not truly reproducible.

Experiment History

2024-05-28 14:32	xgboost_v1.2.0	0.972
2024-05-27 09:11	lightgbm_v1.2.0	0.965
2024-05-26 16:45	catboost_v1.1.0	0.958
2024-05-25 10:02	rf_baseline_v1.0	0.912

Explore the project on GitHub

github.com/FaramarzKowsari/automl-reproducibility-hub

Manifests, Versions & Fingerprints

How AutoML Reproducibility Hub Records Reproducibility Metadata

Every experiment is captured as a manifest — a complete, machine-readable snapshot of data, parameters, environment, metrics, and lineage — and turned into a SHA-256 fingerprint that becomes its unique, tamper-evident identity.

Experiment Manifest (Reproducibility Record)

Seed	42
Dataset Name	Titanic
Dataset Version	v1.2.0
SHA-256 (Dataset)	3f8c...a9d2
Rows	891
Columns	12
Target	Survived
Task Type	Binary Classification
Split	Train: 80% Test: 20%
Preprocessing	Impute(median) + OneHot
Model	XGBoost

Parameters	{max_depth: 6, n_estimators: 200, learning_rate: 0.05, subsample: 0.8}
Metrics	Accuracy: 0.842 ROC AUC: 0.912
Runtime Versions	Python: 3.12.2, scikit-learn: 1.4.2, xgboost: 2.0.3, pandas: 2.2.1
App Version	automl-hub: 1.2.0
Timestamp (UTC)	2024-05-28T14:32:10Z
Experiment Fingerprint	a73fe9b2c0dd8f1e6 98bde2...73b3

Reproducible

From Manifest to Identity

1. Canonical JSON Representation

The manifest is serialized into a canonical JSON (sorted keys, stable formatting).


```
{
  "seed": 42,
  "dataset_name": "Titanic",
  "dataset_version": "v1.2.0",
  "dataset_sha256": "3f8c...a9d2",
  "rows": 891,
  "columns": 12,
  "target": "Survived",
  "task_type": "Binary Classification",
  "split": {"train": 0.8, "test": 0.2},
  "preprocessing": "Impute(median)+OneHot",
  "model": "XGBoost",
  "parameters": {"max_depth": 6,
  "n_estimators": 200, "learning_rate": 0.05,
  "subsample": 0.8},
  "metrics": {"accuracy": 0.842,
  "roc_auc": 0.912},
  "runtime_versions": {"python": "3.12.2",
  "xgboost": "2.0.3", "sklearn": "1.4.2",
  "pandas": "2.2.1"},
  "app_version": "1.2.0",
  "timestamp_utc": "2024-05-28T14:32:10Z"
}
```


2. SHA-256 Hash

A SHA-256 digest is computed from the canonical JSON.

SHA-256

a73fe9b2c0dd8f1e6 98bde2...73b3

3. Stable, Unique Identity

Any change in the manifest produces a new fingerprint.

Unique Experiment Identity

What Goes Into the Fingerprint?

- **Data**: Dataset name, Version, SHA-256, Shape, Target
- **Parameters**: Model, Hyperparameters, Preprocessing, Split strategy, Seed
- **Runtime**: Python version, Library versions, System info, App version
- **Metrics**: Evaluation metrics, Scores, Cross-validation, Details

SHA-256 over canonical JSON → **Unique Experiment Identity**

Why Dataset Versioning Matters

v1.2.0

- Data changes silently over time.
- Versioning + dataset hash guarantee you know exactly which data you used.
- Enables reliable reruns and audits.

Why Hashes Matter

- Hashes are tamper-evident and compact.
- They provide a stable, content-based identity across systems.
- Perfect for storage, indexing, and integrity checks.

How Fingerprints Help

- Quickly compare experiments.
- Detect identical runs instantly.
- Re-run anywhere, get the same result.
- Great for tracking, sharing, and publishing.

Any change to seed, dataset hash, parameters, or runtime versions creates a new identity.

Change Seed | Change Dataset / Hash | Change Parameters | Change Runtime Versions = New Experiment Identity

Explore the project on GitHub

github.com/FaramarzKowsari/automl-reproducibility-hub

Metrics, Comparison & Outputs

AutoML Reproducibility Hub evaluates experiments with standard metrics, compares models side-by-side, and provides rich visualizations and exportable artifacts so results are transparent, repeatable, and easy to share.

Classification Metrics

Accuracy
Overall proportion of correct predictions.

Precision
Of predicted positives, how many were correct.

Recall (Sensitivity)
Of actual positives, how many were found.

F1 Score
Harmonic mean of precision and recall.

ROC AUC
Area under the ROC curve. Threshold-free performance.

Confusion Matrix
Detailed counts of TP, FP, TN, FN.

Regression Metrics

MAE
Mean Absolute Error. Average magnitude of errors.

RMSE
Root Mean Squared Error. Penalizes larger errors.

R^2 (Coefficient of Determination)
Proportion of variance in y explained by the model.

↓ Lower is better ↑ Higher is better

Comparison Dashboard

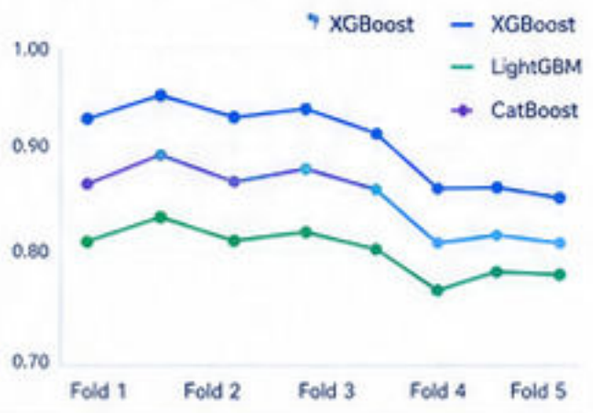
Compare experiments across metrics, datasets, and time.

Top Metric (Primary): ROC AUC | Dataset: All | Models: All | Runs: All | Sorted by: ROC AUC

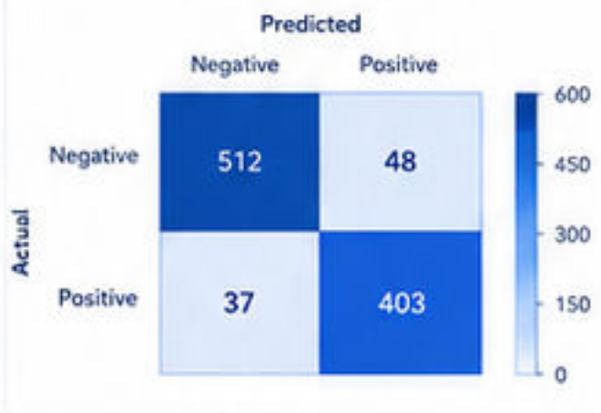
Leaderboard (Top 6)

Rank	Model	Dataset	ROC AUC ↑	F1 Score	Accuracy
1	XGBoost	v1.2.0	0.934	0.912	0.926
2	LightGBM	v1.2.0	0.923	0.901	0.915
3	CatBoost	v1.1.0	0.912	0.889	0.904
4	Random Forest	v1.0.0	0.889	0.862	0.879
5	Logistic Regression	v1.0.0	0.842	0.801	0.817
6	SVM (RBF)	v1.0.0	0.813	0.771	0.789

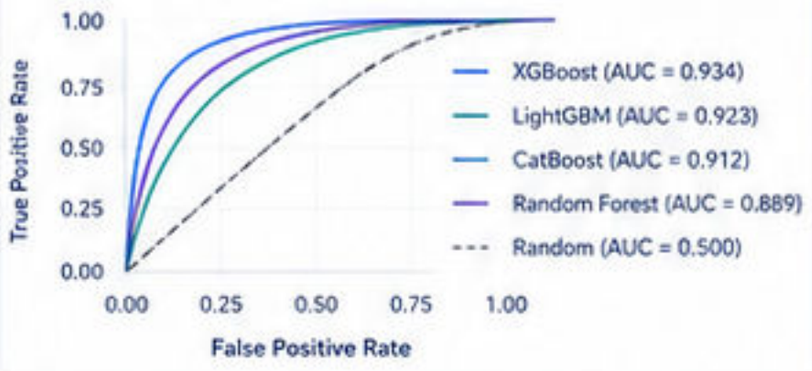
Metric Trends (Cross-Validation)



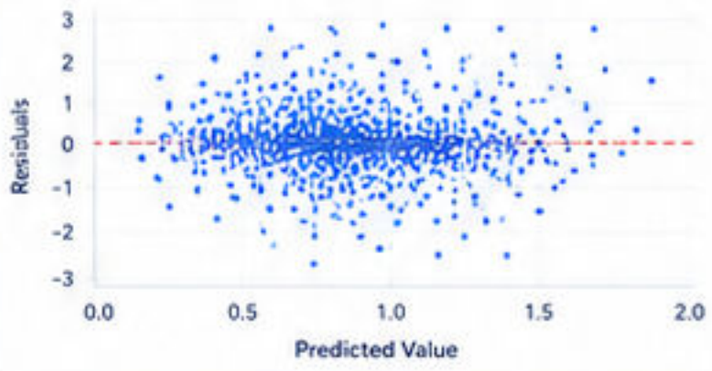
Confusion Matrix (Best Model: XGBoost)



ROC Curves



Residual Plot (Best Model: XGBoost)



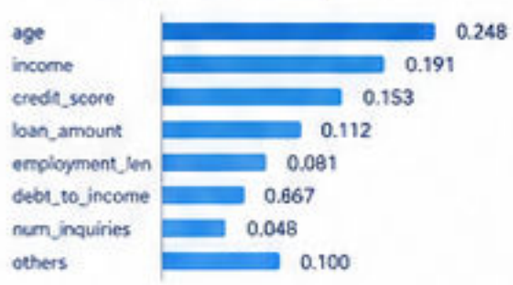
Use the dashboard to quickly identify the best-performing model, understand errors, and make confident decisions.

- Rank by any metric
- Filter by dataset, model, or date range
- Drill into runs for full experiment details
- Export charts and tables

Feature Importance / Coefficient Interpretation

Understand what drives the model's predictions.

Tree-Based (XGBoost)



Linear Model Coefficients (Logistic Regression)

- Positive coefficient/importance increases the prediction.
- Negative coefficient/importance decreases the prediction.
- Magnitude shows strength of impact.

Outputs You Can Export

Experiment Manifest (JSON)
Complete definition of experiment, data, model, metrics, environment.
.json

Result Summaries
Aggregate performance across folds, splits, and runs.
.csv

Metric Snapshots
Time-stamped metric values for tracking and reports.
.json/.csv

Reproducibility Fingerprint
Unique fingerprint of code, data, params, and env.
.fp.json

Python Reproduction Skeleton
Starter script to reproduce the experiment end-to-end.
.py

Reproducibility Score

Excellent

How it works

- Documentation completeness
- Code availability
- Data accessibility
- Environment capture
- Parameter logging
- Metric tracking

Higher scores mean higher trust and easier reuse.
Well-documented experiments are easier to verify, share, and build upon.

Explore the project on GitHub
github.com/FaramarzKowsari/automl-reproducibility-hub

Research Value, Privacy & Deployment

Why This Repository Matters

Transparent • Shareable • Reproducible • Browser-First

Static Mode
Fast - Deterministic
Shareable

Full Mode
Extended Models
Advanced Features

1 Research & Teaching Value

- Reproducible experiments in a transparent environment
- Compare models, metrics, and datasets side by side
- A living resource for lectures, assignments & workshops
- Bridge theory to practice with interactive demos

Central Ecosystem: From Code to Reproducible Impact

Everything is open, versioned, and verifiable.

2 Privacy-by-Design

- 100% browser-local processing
- No data leaves your device
- No analytics, no tracking
- Datasets stay local (uploaded files)
- Share results, not data

Your data. Your device. Your control.

3 Browser-Local Workflow

Load Data (CSV/JSON)
↓
Preprocess
↓
Train & Tune Models
↓
Evaluate & Compare
↓
Export Results (Local)

All computation happens in your browser.
No installs. No servers. No upload.

4 Limitations (By Design)

- **Browser memory constraints**
  Very large datasets may not fit in memory.
- **Full Mode startup is heavier**
  Loads more models and modules for advanced features.
- **Floating-point tolerance**
  Minor numeric differences may occur across environments.
- **Educational transparency**
  Built for learning & research, not for production-scale serving.

GitHub Actions CI

- ✓ Runs on every push & PR
- ✓ TypeScript build & lint
- ✓ Python (Pyodide) tests
- ✓ Unit & integration tests
- ✓ Artifacts & logs for visibility

Status passing

GitHub Pages Deployment

- ✓ Automatic deployment from main branch
- ✓ Live demos & guidebook
- ✓ Fast, global CDN
- ✓ Always up-to-date
- ✓ Custom domain ready

Live Site online

Open-Source Documentation

- ✓ README for quick start
- ✓ Docs for deep reference
- ✓ Inline code comments
- ✓ Architecture & design notes
- ✓ Issue templates for help

Docs available

Citation Readiness

- ✓ CITATION.cff included
- ✓ Zenodo-ready metadata
- ✓ DOI minting in one click
- ✓ Clear authorship & license
- ✓ Reproducible references

CITATION.cff ready

Reproducible Sharing

- ✓ Share results, not data
- ✓ Export metrics & charts
- ✓ Config JSON export
- ✓ Versioned releases
- ✓ Re-run anywhere, same outcome

Reproducible by design

Why This Repository Matters

Empowering reproducible ML experiments for everyone.

Scholars
- Reproduce results with confidence
- Compare methods fairly
- Publish with transparency

Instructors
- Interactive demos for classes
- Assignments with instant feedback
- Lower barrier to experimentation

Developers
- Clean, modular, typed codebase
- Extensible & well-documented
- CI, tests & modern tooling

Students
- Learn by doing
- Explore models visually
- Build real ML intuition

Publication & Citation Ready

README.md — Overview, setup, features, usage → **CITATION.cff** — Standard citation metadata → **Zenodo-Ready** — One-click DOI & metadata → **Guidebook** — Visual guide series (this project)

Cite. Share. Advance Science.

About the Guidebook

Browser-Based ML Stack

TypeScript
Robust Frontend Logic

Pyodide
Python in Web (WASM)

DuckDB-WASM
In-Process Analytics

All running in your browser. No install. No server. 100% local.

Inside AutoML Reproducibility Hub is a ten-page visual handbook that serves as the official guidebook for the GitHub repository AutoML Reproducibility Hub.

This guide explains the project's purpose, architecture, Static mode (S) and Full mode (F), seed control, dataset versioning, parameter tracking, experiment fingerprints, metrics, and reproducibility workflows. It also introduces the browser-based machine learning stack built with TypeScript, Pyodide, and DuckDB-WASM for transparent, shareable, and reproducible AutoML experiments.

Deterministic Randomness | Versioned Datasets | Tracked Parameters | Experiment Fingerprints | Comprehensive Metrics | Reproducible Results

> "From experiment to explanation — reproducible by design."

About the Author

Faramarz Kowsari is an author, Software Engineer and AI researcher based in Istanbul. Focusing on the intersection of technology, education, and personal growth, he has published over 80 digital titles on international platforms. His areas of expertise span Artificial Intelligence, prompt engineering, modern trading strategies (Smart Money Concepts & algorithmic trading), as well as classical literature and mindfulness. In addition to writing, he develops web-based educational tools and creates specialized instructional video content.

Official Profiles

ORCID:	0000-0003-1692-0453
Google Scholar	scholar.google.com/citations?user=FaramarzKowsari
GitHub	github.com/FaramarzKowsari
LinkedIn	linkedin.com/in/faramarzkowsari
Google Books	books.google.com /Faramarz_Kowsari
Personal Website	FaramarzKowsari.github.io
Zenodo	zenodo.org /search?ln=en&cc=Faramarz%20Kowsari

Reproducible by Design

1. **Run Experiment**
   Configure data, models, parameters, and seed.
2. **Capture Fingerprint**
   System, data, parameters, and environment recorded.
3. **Evaluate & Compare**
   Metrics and visual analytics for clear comparison.
4. **Reproduce Anywhere**
   Same setup, same result — on any supported browser.

Official guidebook for:
github.com/FaramarzKowsari/automl-reproducibility-hub