ModelLab Web App Spec (Big Project 0)

1) Product Summary

ModelLab is your personal, portfolio-grade **ML experiment command center** that lets you:

1. **ingest and version datasets**,
2. **run training jobs** (baseline-first),
3. **evaluate with a shared EvalHarness**,
4. **store artifacts + configs + metrics** in a reproducible way, and
5. **compare runs** (diffs, plots, failure slices) so you can iterate fast without chaos.

ModelLab is not a “cool demo.” It is **infrastructure you reuse** for RankForge, TransformerRank, Mini-NeRF, and StateLab. It is the first proof that you ship like an engineer: **reproducible runs, clean evaluation, and honest failure modes**.

---

2) What ModelLab Is For (and What It Is Not)

2.1 Primary Use Cases (v1)

1. **Run tracking**: every train or eval creates a Run with config, seed, commit hash, dataset version, metrics, and artifacts.
2. **Dataset Explorer**: upload a CSV (or connect a dataset folder) and get schema, missingness, basic stats, and slice summaries.
3. **Train and evaluate**: choose a task template (classification/regression) and run:
    1. baseline model
    2. one meaningful improvement
4. **Compare runs**:
    1. metric diffs
    2. config diffs
    3. artifact diffs
    4. “what changed my mind” notes
5. **Export reproducibility pack**:
    1. pinned environment
    2. dataset checksum + version
    3. reproduce command
    4. artifact bundle path

2.2 Non-Goals (v1)

1. Not a full hosted ML platform like SageMaker
2. Not full-scale distributed training
3. Not a generic notebook replacement
4. Not a general purpose ETL warehouse

2.3 Core Promise

If someone clones your repo and follows the README, they can reproduce:

1. one dataset inspection
2. one train run
3. one eval report
4. at least one plot
5. at least one failure slice
6. a run comparison

---

3) Core Concepts and Mental Model

ModelLab revolves around 6 objects:

1. **Project**
- a workspace for a specific modeling effort
- contains datasets, runs, models, reports
2. **Dataset Version**
- a dataset plus metadata: checksum, schema snapshot, split strategy, provenance, and notes
3. **Run**
- a single execution of train or eval
- stores config, seed, code version, metrics, artifacts, logs
4. **Model Artifact**
- the saved model weights and metadata (plus inference signature)
5. **Eval Report**
- a structured output from EvalHarness (metrics + plots + slices + failure examples)
6. **Repro Pack**
- a portable bundle of everything needed to reproduce a run

Everything in the UI is basically: **Datasets** → **Runs** → **Reports** → **Comparisons**.

---

4) EvalHarness (Deep Context, Purpose, and Design)

4.1 What EvalHarness Is

EvalHarness is a shared evaluation library and protocol that enforces **honest, repeatable, comparable evaluation** across every project you ship in 2026.

If ModelLab is your “experiment cockpit,” EvalHarness is the **instrument panel calibration standard**. It makes sure your results are not vibes.

EvalHarness is:

1. **a code module** (Python package inside your repos)
2. **a standard output format** (metrics, plots, slices, failure examples, takeaways)
3. **a checklist of required eval behaviors** (baseline-first, no leakage, slicing, CIs, stress tests)

EvalHarness exists so that:

- **all projects produce comparable artifacts**
- you do not rewrite evaluation logic differently in each repo
- every “result” is tied to a dataset version, config, and reproducible run
- you build a muscle memory of evaluation rigor

4.2 The Problem EvalHarness Solves

Without a harness, evaluation becomes:

- inconsistent metric choices
- missing baselines
- silent leakage
- no uncertainty reporting
- cherry-picked examples
- plots that are not reproducible
- “it worked” without failure analysis

EvalHarness prevents that by forcing:

1. **standard metrics** per task type

2. **standard slice reports**
3. **standard failure example selection**
4. **standard artifact saving paths**
5. **standard run metadata capture**
6. **standard stress tests and degradation reporting**

This is one of your strongest “actually smart company” signals because it demonstrates:

- you care about **measurement**
- you care about **reliability**
- you care about **truthful claims**
- you know how to build systems that scale across projects

4.3 What EvalHarness Outputs (Contract)

Every evaluation produces a **single folder** that is “the evaluation artifact.”

4.3.1 Required Files

1. `eval_summary.json`
- high-level metrics, key plots, dataset version, run id, commit hash
- pointers to other files
2. `metrics.json`
- task metrics
- must include baseline metrics when applicable
3. `confidence_intervals.json` (nullable but preferred)
- bootstrap CIs for selected metrics
- includes method details (bootstrap count, seed)
4. `slices.json`
- metrics computed on slices
- each slice includes:
    - slice definition
    - sample size
    - slice metrics
    - delta from overall
5. `failure_examples.json`
- a stable selection of examples that show what broke
- includes:
    - example id
    - input features summary
    - prediction

    - ground truth
    - model confidence if applicable
    - why it is a failure
    - optional link to raw row or artifact
6. `takeaway.txt`
- exactly 5 sentences
- forces you to compress signal without hand-waving
7. `plots/`
- png (or svg) plots, deterministically generated
8. `repro.md`
- a short reproduce snippet:
    - install command
    - dataset retrieval
    - run command
    - eval command

4.3.2 Naming and Paths

EvalHarness enforces:

- `/artifacts/<project>/<run_id>/eval/`
- consistent naming so a compare tool can locate artifacts automatically

---

4.4 What EvalHarness Computes (Standardized)

EvalHarness has task-specific evaluator modules.

4.4.1 Classification Evaluator

Metrics:

1. accuracy
2. ROC-AUC (binary where valid)
3. PR-AUC (binary where valid and imbalanced)
4. F1 (macro or weighted depending on class counts)
5. confusion matrix

Plots:

1. confusion matrix heatmap
2. ROC curve (if binary)
3. PR curve (if binary)

4. calibration curve (optional but strong)
5. prediction confidence histogram

Slices (default set):

1. predicted confidence deciles
2. missingness bucket (low/medium/high missingness)
3. a key categorical feature bucket (top 5 categories)
4. class-specific slice (each class, or top 3 frequent classes)

Failure example selection rules:

1. top N confident wrong predictions
2. top N low-confidence correct predictions (model uncertainty edge cases)
3. top N errors on the worst-performing slice

Stress tests (v1 lite, still valuable):

1. missing value corruption (mask some fraction)
2. label noise injection (small percent, just for sensitivity)
3. feature corruption (shuffle one column)
   Outputs:

- degradation table and one plot

4.4.2 Regression Evaluator

Metrics:

1. MAE
2. RMSE
3. R^2 (optional, but fine)
4. median absolute error (optional, good robustness signal)

Plots:

1. residual plot
2. predicted vs actual scatter
3. error distribution histogram
4. error vs target quantile bucket

Slices:

1. target quantiles (0–20, 20–40, etc.)
2. a key feature bucket

3. missingness bucket

Failure example selection:

1. top N largest absolute errors
2. top N underprediction errors
3. top N overprediction errors

Stress tests:

1. noise injection on selected features
2. missing value corruption
3. distribution shift simulation (train on subset, test on held-out segment)

4.4.3 Ranking and Retrieval Evaluator (future reuse in RankForge)

This is important because ModelLab will grow into RankForge support.

Metrics:

1. Recall@K
2. MRR@K
3. NDCG@K
4. latency measures (candidate generation and reranking)

Plots:

1. metric vs K curves
2. recall vs latency tradeoff plot
3. slice performance bars

Slices:

1. query length bucket
2. head vs tail queries
3. entity density bucket
4. OOD query patterns (if you define them)

Failure examples:

- queries where ranking fails with top retrieved docs shown

Stress tests:

- corrupted queries

- missing fields
- noise in embeddings
- ANN index parameter shifts

4.5 The “No-Leakage” and “Baseline-First” Enforcement

EvalHarness is not passive. It enforces rules.

4.5.1 Leakage Prevention

EvalHarness requires the pipeline to expose:

- train split indices
- val/test indices
- preprocessing fit objects

It checks:

1. preprocessors were fit only on train
2. no test rows appear in train
3. time-based splits preserve ordering if declared time series
4. categorical encoders do not peek at full data

If a check fails:

- eval fails hard
- run status becomes failed with reason

4.5.2 Baseline Enforcement

EvalHarness requires:

- baseline model metrics exist before improved model metrics count as meaningful

Meaning:

- you cannot claim improvement without a baseline artifact
- compare view always shows baseline vs improved

4.6 Confidence Intervals (Practical, Not Academic)

EvalHarness includes bootstrap CI utilities:

- choose metric
- resample test rows with replacement
- compute metric distribution
- save:
    - mean
    - 2.5th and 97.5th percentiles
    - number of bootstraps
    - seed

Rules:

1. you do not need CIs for every metric, but you must pick “headline metrics”
2. you must record bootstrap count and seed
3. if compute is heavy, use smaller bootstraps but disclose it

---

4.7 Run Comparability and “Compare Mode”

EvalHarness is designed for comparisons.

4.7.1 Comparison Inputs

A compare tool can take:

- two run ids
- load their eval artifacts
- compute:
    - metric deltas
    - slice deltas
    - failure category diffs

4.7.2 Comparison Outputs

1. “What improved overall”
2. “What got worse”
3. “Which slices shifted”
4. “Which failures are new”
5. “What changed in config”
6. “How confident we are” (via CI overlap notes)

This is the thing that makes your work feel like:

- engineering, not experiments
- disciplined iteration, not random tuning

4.8 Failure Modes Taxonomy Integration

EvalHarness auto-generates a Failure Modes template per run:

1. Data failures
2. Objective/metric failures
3. Model/representation failures
4. Optimization failures
5. Systems/infra failures
6. Human/UX failures (if applicable)

It also supports linking failures to examples:

- each failure example can be tagged with a failure mode category
- compare mode can show “failure mode distribution changed”

4.9 Latency and Performance Harness (v1-lite but real)

EvalHarness includes a small benchmarking helper:

- measure inference latency with warmup
- record p50 and p95
- record batch size and device
- record time per sample

This matters later for:

- RankForge serving
- StateLab throughput
- general systems credibility

4.10 EvalHarness Library Structure (Implementation Design)

Inside `/ml/evalharness`:

1. `core/`
- common interfaces
- artifact writer
- schema definitions
2. `metrics/`
- metric functions and wrappers
3. `plots/`
- deterministic plotting utilities
4. `slicing/`
- slice definition DSL
- slice evaluation runner
5. `failures/`
- failure example selectors
- tagging utilities
6. `stress/`
- corruption generators
- degradation reporting
7. `ci/`
- bootstrap utilities
8. `bench/`
- latency and runtime profiling utilities
9. `schemas/`
- pydantic models for outputs so they stay stable

---

5) System Requirements

5.1 Functional Requirements (v1 must-haves)

1. **Auth**: Google login or magic link email
2. **Projects**: create + list projects
3. **Datasets**:
    1. upload CSV
    2. compute checksum
    3. infer schema
    4. basic profiling (missingness, distributions, label balance)
    5. define a split strategy
4. **Training**:

    1. choose a template (tabular classification, tabular regression, text classification optional)
    2. run baseline
    3. run improved model
    4. capture logs
5. **Evaluation**:
    1. standard metrics
    2. bootstrap confidence intervals (where relevant)
    3. at least one slicing report
    4. at least one failure example set
6. **Artifacts**:
    1. save model
    2. save plots
    3. save predictions sample
    4. store in a consistent /artifacts structure
7. **Run history**:
    1. list runs per project
    2. run detail page
8. **Run compare**:
    1. diff metrics
    2. diff configs
    3. diff dataset version
    4. side-by-side plots
9. **Repro**:
    1. display reproduce command
    2. show env and seed
    3. show dataset checksum
    4. show commit hash

5.2 Non-Functional Requirements (v1)

1. **Reproducibility first**: a run without dataset version + seed + code version is invalid
2. **Fast**: UI should stay responsive while training runs happen
3. **Auditability**: no silent changes, everything is logged and versioned
4. **Secure defaults**: RLS policies, input validation, secrets not in repo

---

6) Supported Templates (Model Tasks)

Start small, professional, and reusable.

6.1 Template A: Tabular Classification (v1 core)

- Baseline: DummyClassifier (most frequent)
- Baseline 2: LogisticRegression
- Improvement: XGBoost or LightGBM
- Metrics, slices, failures, stress tests: via EvalHarness Classification Evaluator

6.2 Template B: Tabular Regression (v1 core)

- Baseline: mean predictor
- Baseline 2: LinearRegression
- Improvement: XGBoostRegressor
- Metrics, slices, failures, stress tests: via EvalHarness Regression Evaluator

6.3 Optional Template C: Text Classification (v1 optional)

- Baseline: TF-IDF + Linear model
- Improvement: small transformer fine-tune
- Metrics, slices, failures: via EvalHarness

If you want, next I can add:

1. The exact **JSON schemas** for `metrics.json`, `slices.json`, and `failure_examples.json` so the contract is rigid.
2. A “golden demo dataset” recommendation list that makes ModelLab look instantly legit.