Beyond the Black Box: *How Intuit Credit Karma Runs ML Explainability for 140M Members with Beam*

Beam Summit 2026


OUR MISSION
Champion financial progress for all.
intuit
credit karma

Help members make smarter money moves, every day.

We go **beyond credit scores** to give members the tools and confidence to take control of their financial lives.

Every product we build is designed to deliver clear value and put members in control of their financial journey.

Money Management

- Cash flow
- Spending
- Expenses
- Tax

Marketplaces

- Cards
- Loans
- Insurance
- Karma Guarantee

Credit Building

- Credit Monitoring
- Credit Spark
- Credit Builder

Credit Karma Money

- Spend
- Save
- Access to Cash

Key Challenges

- **Scale**: Processing 10-100 terabytes of data from various sources daily.
- **The goal**: Turning raw financial data into actionable, real-time advice.

The Usecase

Demystify Model for consumers

- Large Scale Explainability
  - 1+ Billion instances daily with 50000+ attributes (500+ models)
- ML use-cases
  - Explain Offer Badges for members
  - Explain Refund Loan for Tax Filers

Offer Badge Explainability

What is Model Explainability?

- Explainability in machine learning means that you can explain what happens in your model from input to output. It makes models transparent and gives us insight into predictions.

- Local Explanations is what our stakeholders/members are looking for

- Explainability Use Cases:

  Business stakeholders - why did the model score this member this way?
  Data Science - feature attribution for diagnostics, drift detection, model improvement
  Regulators - consumer-facing decisions must be explainable (FCRA, ECOA)
  End users - what's actually affecting my financial outcomes?

Explainability in Dev vs Production — The Gap

1. ML teams face a silent gap between explainability in development and explainability in production.

2. Variety of Tools available : LIME, SHAP, Cloud Managed Explainability etc

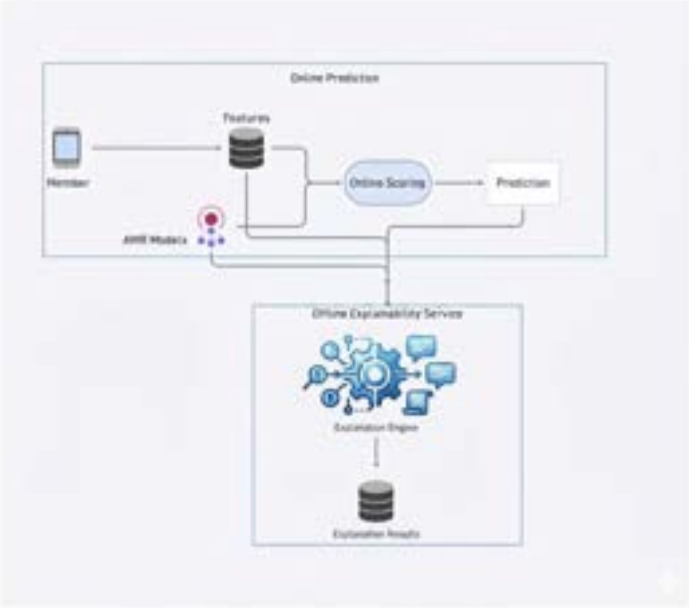
Managed Explainability

3. In notebooks: Explainers runs in seconds, easy to inspect — the problem feels solved

   In production systems we run into several challenges:
   - a) Scalability issues
   - b) Model version drift
   - c) Feature version drift
   - d) Latency
   - e) Observability & Silent failures

Paradigm 1: Post-Hoc Explainability

- Model scores in production first
- Separate batch job reads historical prediction logs from your analytics / warehouse store, runs explainability, and writes explanations back



Where this fits:

- Regulatory & compliance — Reproducible “why” tied to logged decisions (model, inputs, time).
- Protect production scoring — Keep scoring fast and simple; explain off the request path.
- Cost — Batch/offline explain is cheaper than doing heavy work on every live call.
- Richer methods — You can afford slower, more expensive explainers when you’re not on the latency-critical path.

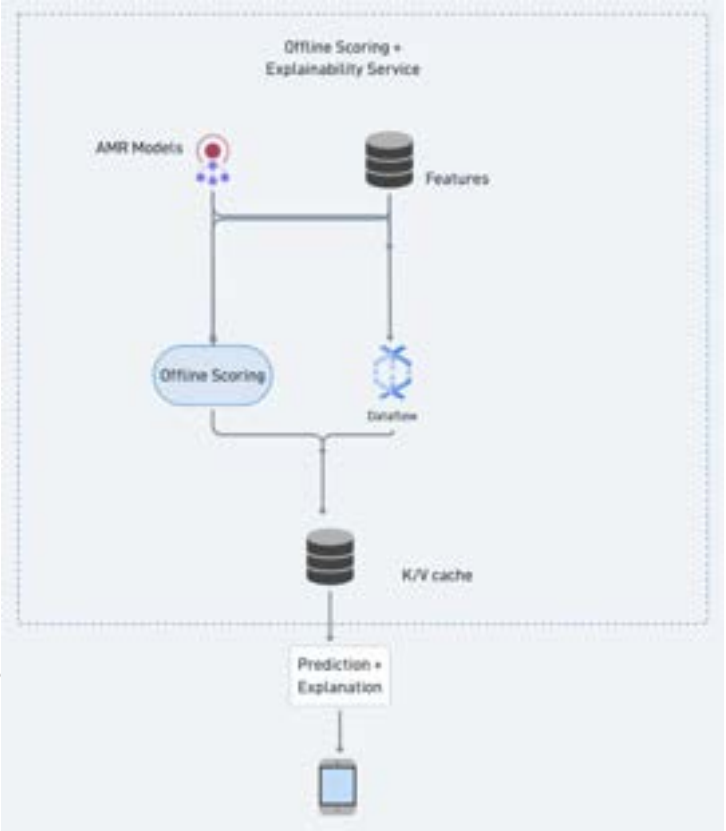
Paradigm 2: Precomputed Explainability

Explain predictions before they are surfaced

- Explanations and scores generated in a batch job for entire population
- When member comes online, we hit the cache/ key-value store for an instant lookup

Where this fits:

- Scoring on the entire population is acceptable as the ratio of daily active members to entire population is significant.
- Cost — Batch/offline explain is cheaper than doing heavy work on every live call.
- Richer methods — You can afford slower, more expensive explainers when you're not on the latency-critical path.



Paradigm 3: Real Time Explainability

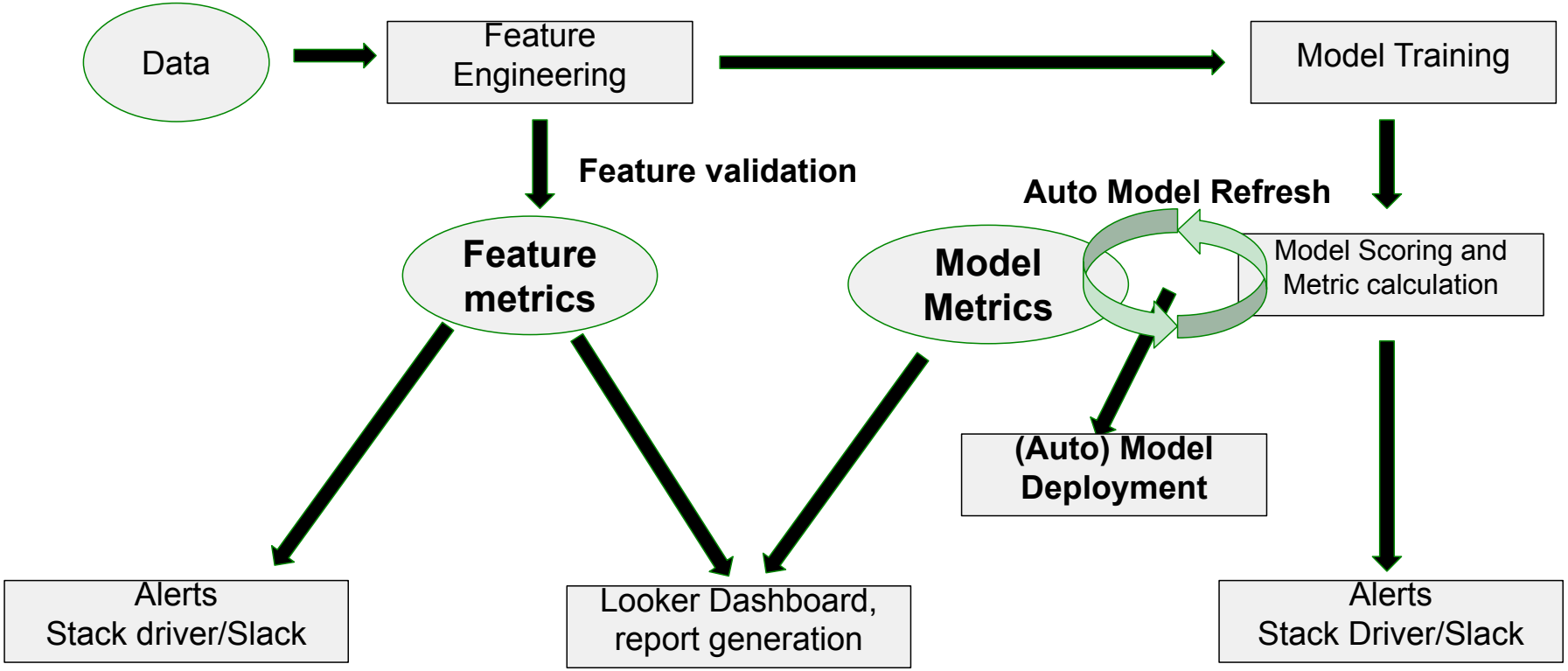
Explain predictions at serving time

The ideal — explanation generated in the same request as the prediction

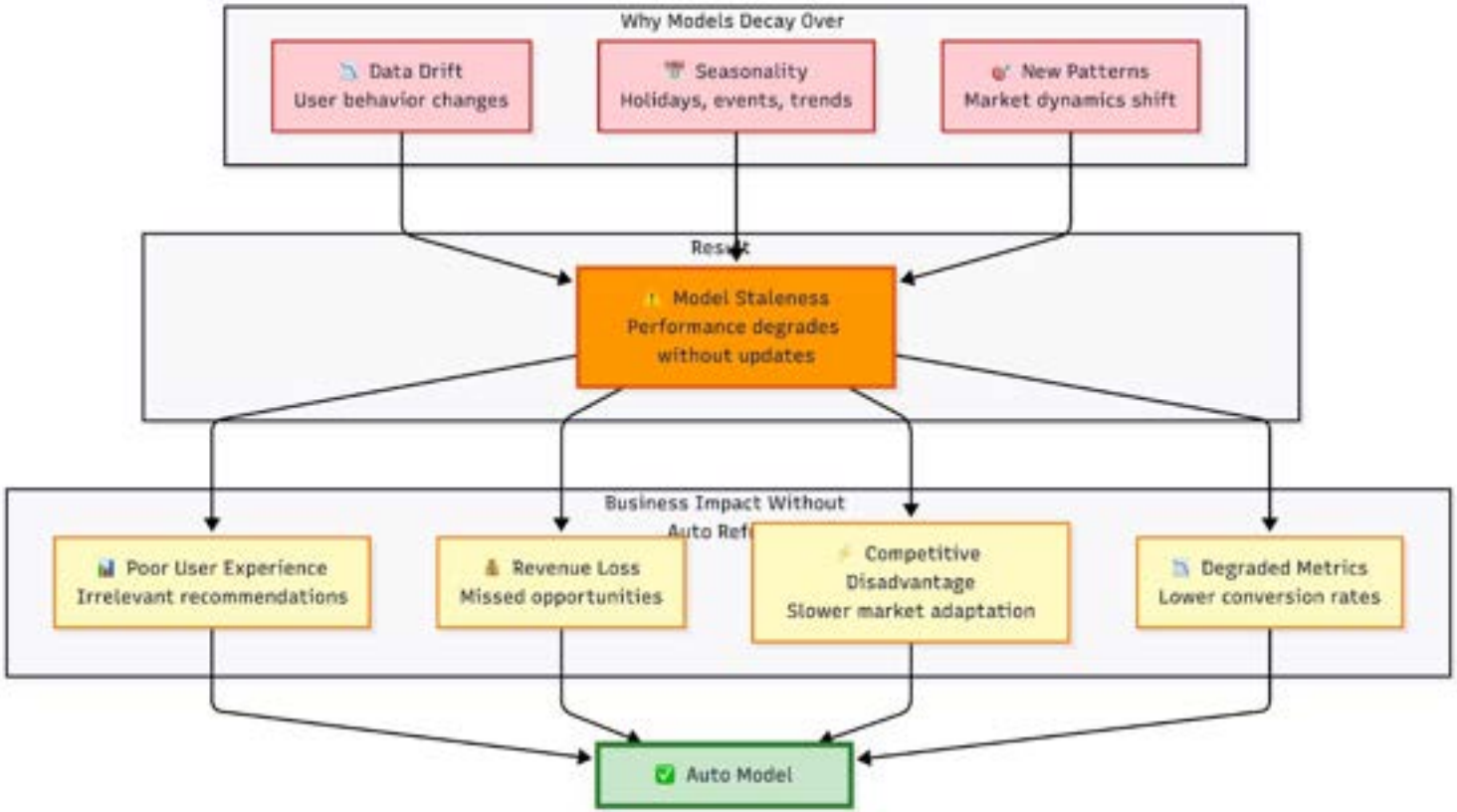
Where this fits:

- Loss in latency is acceptable as explainers are considerably more expensive than model scoring
- Model scoring environment supports Python SDK
- Features shift fast and inaccurate results are not acceptable
- The daily users are a small fraction of the entire population and its overkill to score the entire population

What is Automatic Model Refresh ?



Auto Model Refresh



Tech Stack for ML explainability

Evaluating options for production-scale explainability

Platform requirements for ML explainability as a service

- **Single architecture**
- Onboarding a new model = config exercise, not an engineering project

  Each Data Science team provides:

  1. BigQuery feature table
  2. Code config file
- The platform manages:

  Version fidelity

  Horizontal scaling

  Inline validation

Options Evaluated

1. **Cloud Provider XAI**

   Cloud Provider XAI - We also found this less extensible and customizable for our use case

2. **Custom endpoint**

   Managed Ops, always-on cost, scaling challenges

3. **Apache Beam on Cloud Dataflow** - CHOSEN!

Why Beam Wins : Four Pillars

1. **Ownership & Extensibility**

   Any attribution method, any model format - not constrained by external API surface

2. **Debuggability**

   Full visibility on failures and intermediate outputs

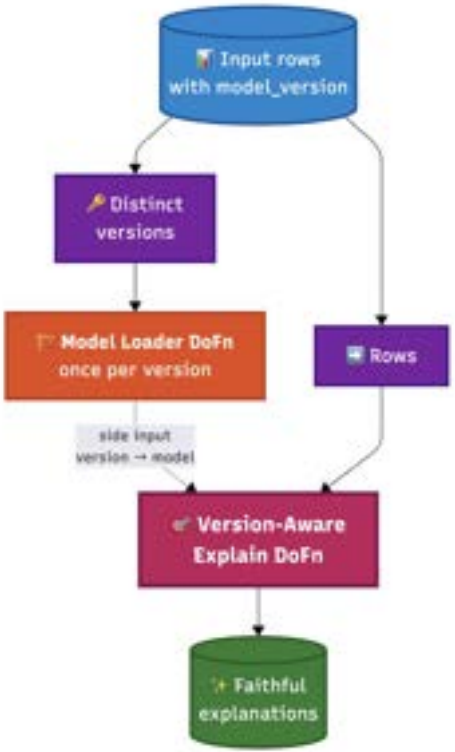
3. **Economics**

   On demand compute - zero idle cost

4. **Support for all 3 paradigms of explainability**

   Post-hoc and precomputed are batch pipelines while real time is a streaming job

Key pattern: Version Aware DoFn



This Version Aware DoFn scales for all three paradigms:

1. Post-hoc Explainability (Batch)
2. Pre-computed Explainability (Batch)
3. Real time Explainability (Streaming)

In the Explain DoFn we can plug any Explainer Kernel

1. TreeShap Explainer
2. KernelShap Explainer
3. LIME explainer
4. TF integrated gradients

Plug different Explainer Kernels

Explainer Kernels can be grouped into two categories:

1. White-box Explainer (needs model structure)

- Tree Explainer (SHAP for trees)
- TF Integrated Gradients (IG)

2. Model-agnostic - Black Box Explainers (treat model as predict)

- LIME — Fits a simple local model (often linear) in a neighborhood around one prediction, using perturbed inputs
- Kernel Explainer (Kernel SHAP) - Approximates Shapley values by solving a weighted regression over perturbed coalitions of features.

Scaling Tips and Tricks

1. **Batching**

   Batching rows before we explain predictions lets Beam do one chunk of work at a time instead of millions of tiny steps.

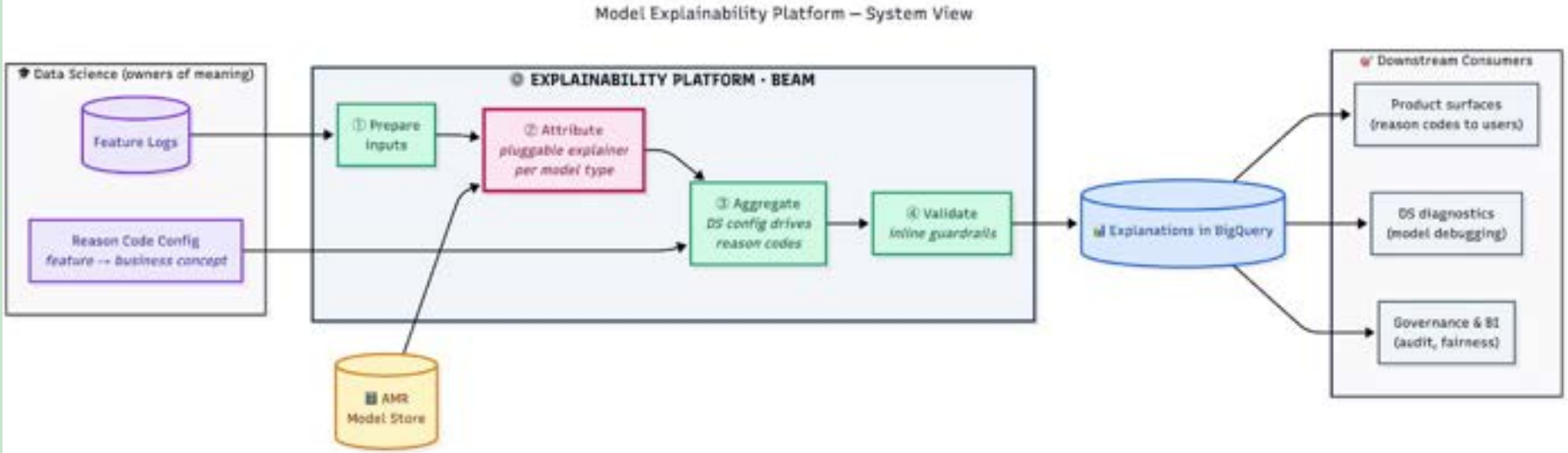
   beam.BatchElements improves latency by 93% compared to row-level explanations

2. **FastTreeShap** - Python library alternative to Shap TreeExplainer but faster!

   TreeExplainer has an optional approximate parameter — set approximate=True for faster, less exact explanations

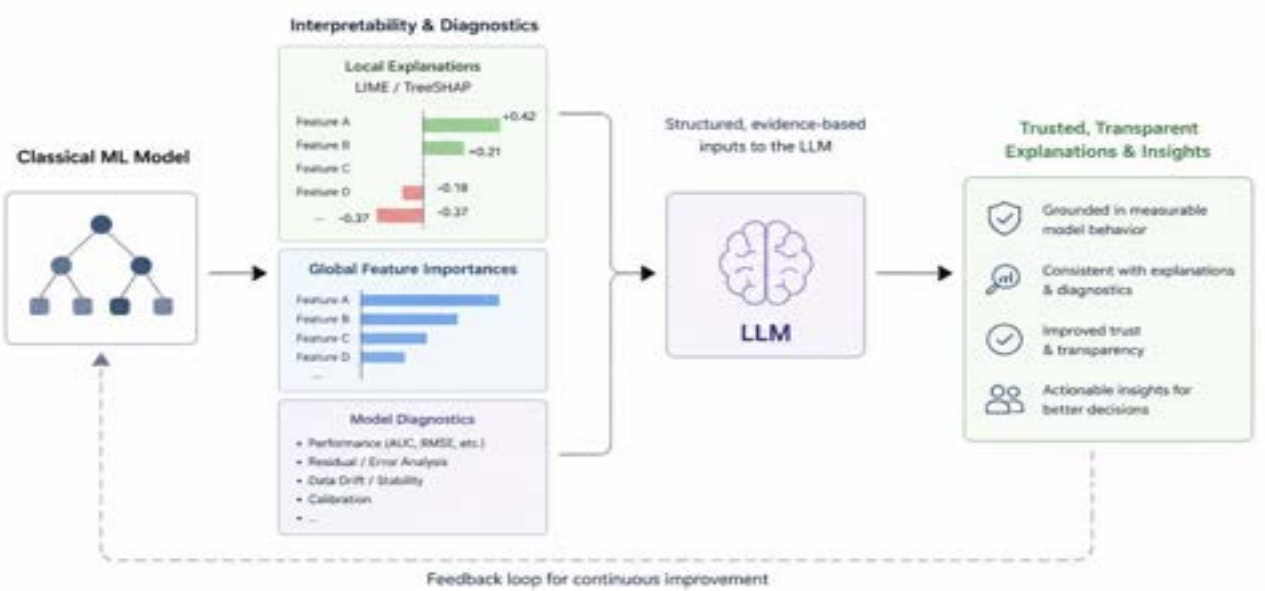
3. **Reuse Previous Results** - if the features and models have not changed - you don't need to generate explanations again!

Architecture Diagram for DS/Config Onboarding



Generating Human Readable Explanations

- Root explainability in classical ML interpretability methods and model diagnostics, and use outputs like LIME/TreeSHAP and feature importances as structured inputs to the LLM
- Improve trust and transparency by ensuring outputs stay aligned with measurable model behavior



Thank You !