Turbox Tax Refund Status Problem

Problem

Today, TurboTax users have little visibility into *when* their refund will arrive or what happens after they e-file their returns.

Once the filing is complete, the product experience essentially ends, leaving customers to check multiple external systems to understand progress.

This lack of visibility erodes trust — users feel uncertain whether their return was accepted, processed, or paid out.

Our goal is to provide accurate, timely, and explainable refund status updates and data backed Estimated Time of Arrival (ETA) for their refunds directly within TurboTax, eliminating the need to visit the IRS or their bank.

For this scope, we are focusing only on **e-filed returns** (amended returns will follow a similar path later).

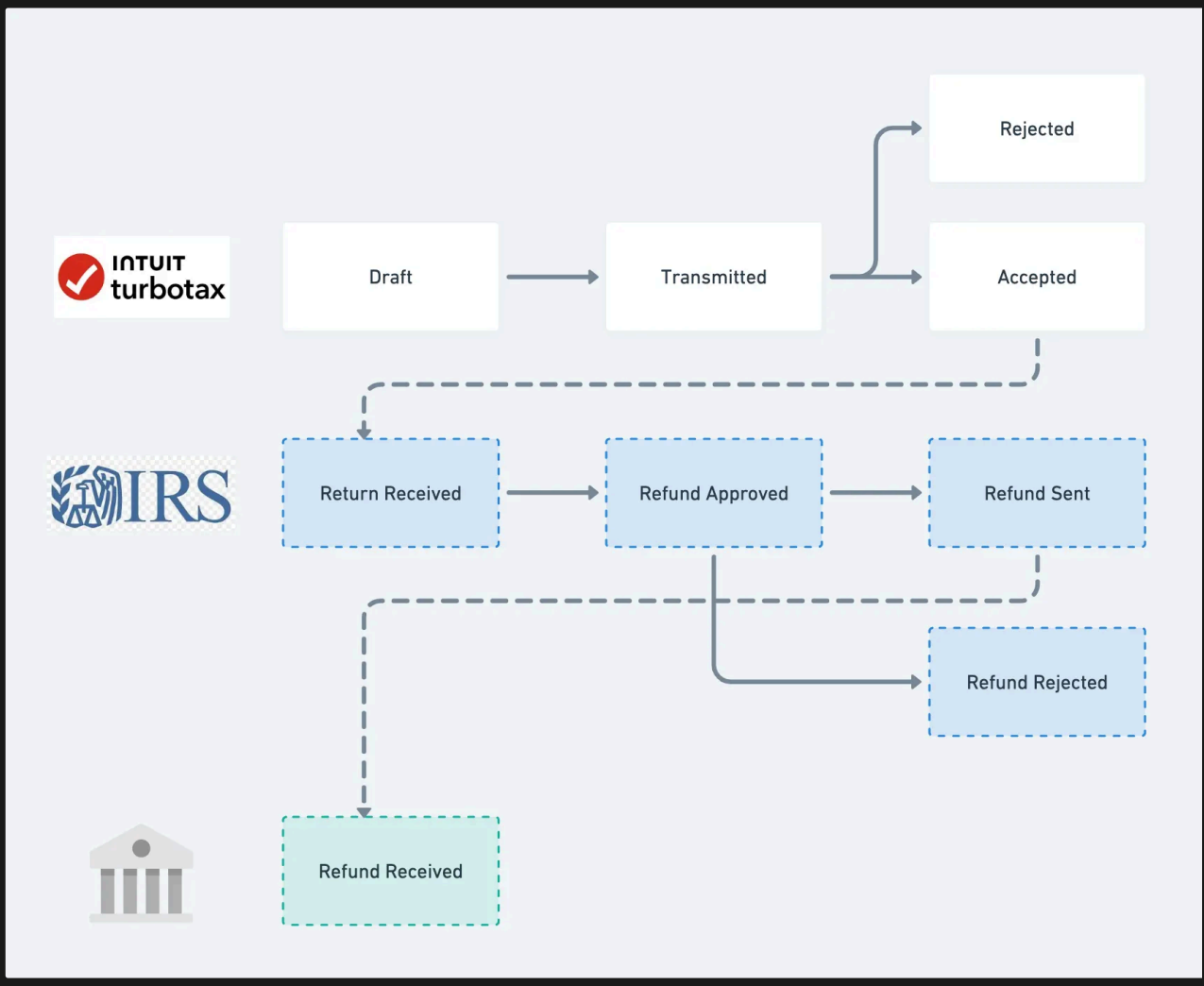
Background / Context

The refund lifecycle today spans several independent systems:

1. **TurboTax** records early states like *Filed* and *Accepted by IRS*.
2. **IRS** maintains refund processing and payment progression through *Where's My Refund*.
3. **Banks / Mint** confirm deposit transactions once the refund is disbursed.

Each of these systems holds only part of the picture, forcing users to check multiple sources to understand refund progress.

The following diagram illustrates these fragmented states and the current handoff between TurboTax, IRS, and the user's bank:



This highlights why a unified "single-glass-pane" experience inside TurboTax is necessary — to merge these disparate sources and add a predictive ETA layer.

Assumptions

- **Existing TurboTax Clients and Auth Flow**

  We will leverage the existing TurboTax client applications (web, desktop, and mobile) for authentication and user interaction.

  Users will continue logging in with their existing Intuit credentials (SSO, MFA, etc.), ensuring no new onboarding friction.

- **Familiar User Experience**

  Customers will continue using the same TurboTax interfaces they are already familiar with — preserving established navigation patterns and reducing cognitive load.

  Enhancements will appear as **progress indicators, ETA estimates, and explanations** within the same UI surfaces (e.g., existing filing progress bar).

- **Incremental Enhancement**

  The solution is designed as an enhancement to existing TurboTax and IRS integration flows, not a separate standalone experience.

  This approach minimizes client-side rework and accelerates rollout across platforms.

- **Data Access and Consent**

  Users will explicitly grant consent for accessing their IRS refund data and (optionally) their bank transaction data via Mint or direct connection.

  These permissions will follow existing Intuit OAuth scopes and consent patterns.

  Example:

  Current TurboTax progress bar ends at Refund Delivery

ref: ✅ why does it say refund delivery instead of telling me my exp...

  This view will be enhanced to show more states and details.

Functional Requirements

- **Post-Acceptance Status Tracking**

  The system must display all refund statuses that occur after IRS acceptance — including *Return Received*, *Refund Approved*, *Refund Sent*, and *Refund Received*.

- **Refund ETA Prediction**

  Users should see a predicted refund arrival date by leveraging historical and current data.

  Each ETA must include a confidence score to help set expectations clearly.

- **Refund Deposit Visibility (Optional)**

  Users who opt in should be able to view refund deposit confirmations sourced from Mint or directly connected bank accounts.

- **Notifications on Status Change**

  The system must emit reliable *status change events* whenever a refund transitions between states. These events will contain complete, deduplicated, and versioned information so they can be consumed by downstream systems responsible for notification delivery (in-app, email, or push).

  This service's responsibility ends at **generating correct and consistent change events**, not performing the last-mile push itself.

- **Explainable ETA**

  Users should have access to an optional "Why this ETA?" view that provides a short, plain-language explanation generated via the LLM

- **Consent and Data Access**

  The system must obtain and persist explicit consent before accessing IRS or banking data on behalf of the user.

  Consent flows will reuse existing Intuit OAuth and permissions infrastructure.

Non Functional Requirements

- **Availability**

  Target availability is **99.99%** during the tax season (January–April) and **99.9%** otherwise.

  Assumption: Limited downtime (~5 minutes/month) is acceptable since this is not part of the filing submission workflow.
- **Latency**

  p90 latency for `GET /status` should be under **1 second**
- **Scalability**

  The system should handle up to **100 million refund status events** per month with bursty nature
- **Security**

  All data in transit and rest must be encrypted
- **Privacy and Compliance**

  Must adhere to IRS Safeguard requirements and Intuit's internal privacy and data-handling policies.

  Access to user financial data is strictly scoped and auditable.
- **Reliability and Graceful Degradation**

  In case of replica lag or cache outage, the system should serve slightly stale data rather than fail the request.

  Change Notifications / Webhooks must not fire until cache and database states are consistent. So a GET call after Webhook firing should reflect latest changes communicated by Webhooks.
- **Transparency and Trust**

  The system must communicate confidence levels and reasons for delay clearly to reinforce user trust.

  ETAs and explanations must always be traceable back to real data or model outputs.

Out of Scope

- **Last-Mile Notification Delivery**

  The service is not responsible for sending in-app, email, SMS, or push notifications to end users.

  It only produces accurate and versioned change events that downstream systems can consume for delivery.
- **Amended and Paper Returns**

  This design focuses exclusively on e-filed returns. Amended e-files will be a future follow up. Paper filings is not planned.
- **Payment Routing or Refund Disbursement**

  The system will not initiate or manage any financial transactions. It only reflects refund status changes provided by IRS or linked banking data.
- **Client-Side UI Implementation**

  While the design proposes enhancements to the TurboTax progress bar and ETA display, client-side UI work (web, desktop, or mobile) is owned by the respective TurboTax product teams. We propose some changes as an assumption so we can construct APIs accordingly.
- **AI Model Development and Training**

  The design assumes availability of a trained regression model and LLM explanation service. Model training, feature engineering, and evaluation pipelines are owned by the **AI Platform team**. We make assumptions on AI pipelines to ensure we have correct data models, system and infrastructure in place to support those AI pipelines.

- **User Identity or Authentication**

  Authentication, authorization, and session management are handled by existing Intuit identity systems (SSO, OAuth, MFA). This service only consumes validated identity tokens. It also assumes a middleware which will validate the tokens and then call this service. Essentially, this service dont get invalid token calls.

- **Manual Support Workflows**

  The system does not interface with customer support tools or manually override refund statuses. All updates must originate from IRS, Mint, or automated pipelines. However, the plan is to forward context in Support tools so Customer Support is aware of these changes.

- **Analytics**

  We can perform various analytics like how many users got approved on X day , within X days. This is considered out of scope. If we have to pull in, I would implement a CDC like replication to a OLAP store and run all my analytics there. For now keeping design focused on main content.

Solution Overview

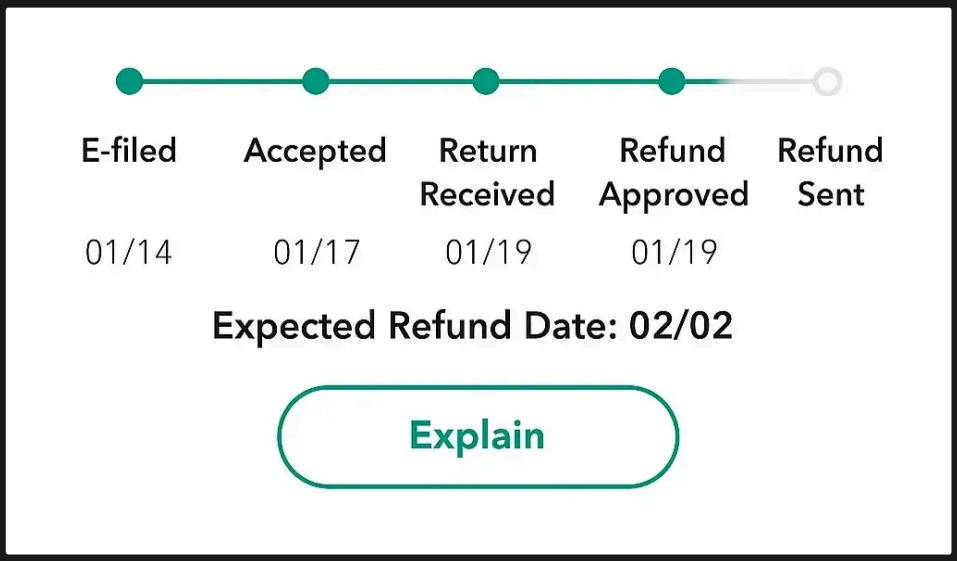
The goal of this system is to make TurboTax the **single, trusted source** for a filer's refund journey — combining IRS status data, banking deposit confirmations, and machine-learned refund ETAs into one consistent and explainable experience.

At its core, the design introduces a **Refund Status Service** that listens to events from IRS and banking integrations, computes predicted ETAs through an AI pipeline, and surfaces the unified refund timeline back to TurboTax clients.

The system focuses on producing accurate change notifications for downstream services (notifications, UI, analytics) which can be leveraged for tighter customer interaction.

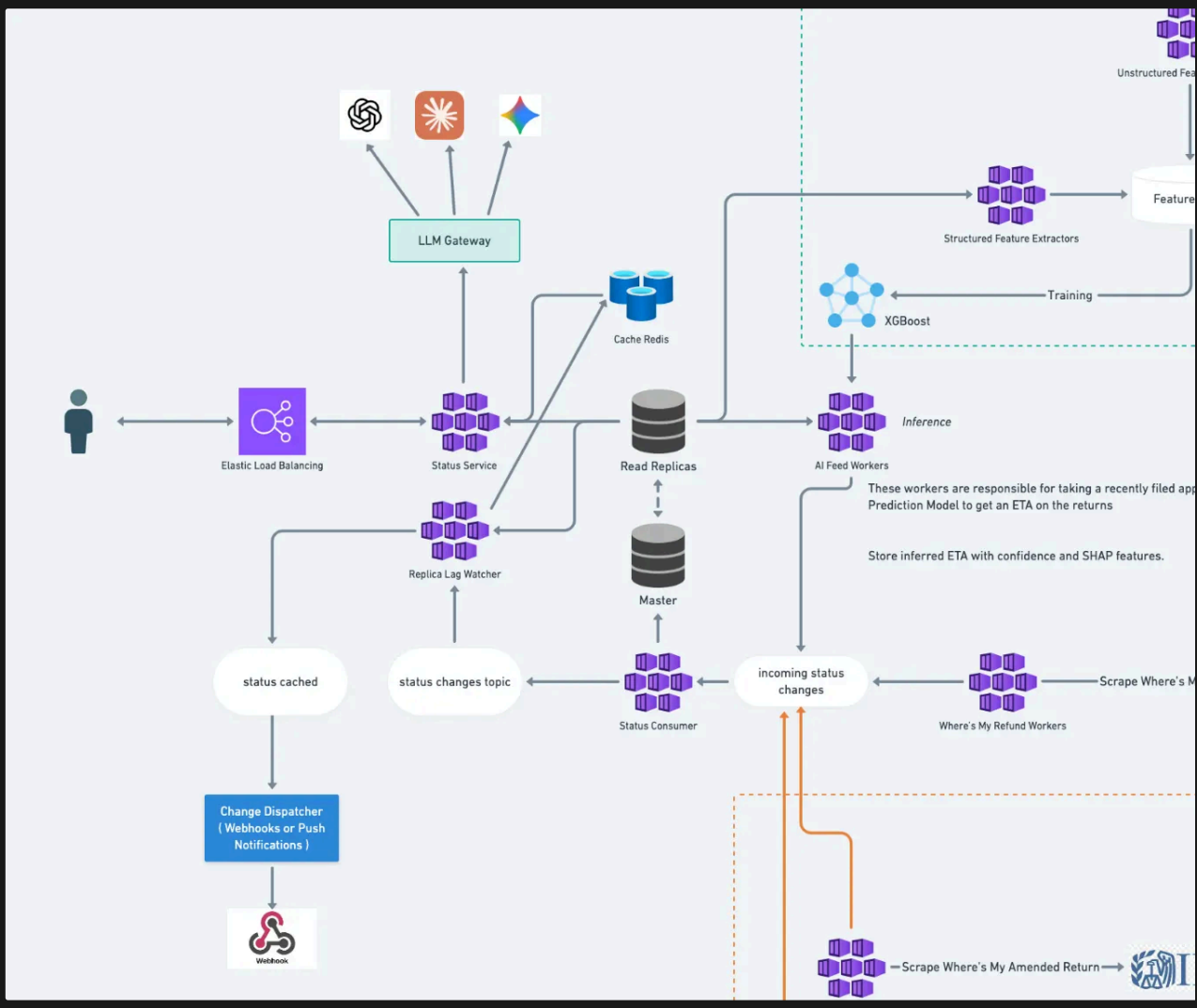
Expected Outcomes

- TurboTax users gain continuous visibility into refund progress with accurate ETAs and clear explanations.
- All systems (IRS, Mint, and TurboTax) appear as a single, coherent timeline, reducing uncertainty and customer support contacts.
- The architecture scales linearly with traffic peaks during tax season and can extend easily to amended returns or new data partners in the future.



High Level Design

The Refund Status Service is an **event-driven, replication-aware system** that ensures refund status updates are always correct and consistent before being exposed to users or downstream systems.



Detailed Design

Components

- **Status Consumer**
  - Validates and writes updates from IRS, AI, or Mint sources.
  - Performs monotonic status validation (no regressions).
  - Updates the `returns` table and appends the new status to the `history` JSON array.
  - Publishes a lightweight event to the `status_changes` Pub/Sub topic after commit.

- **Replica Log Watcher**
  - Subscribes to the `status_changes` topic.
    - Sample payload

      ```
      { "return_id": "01J6WFCQFPJ5B0CBRHEPQK4AAG", "status": "APPROVED",
      "updated_at": "2025-02-06T15:45:00Z", "eta_date": "2025-03-20",
      "confidence": 0.94 }
      ```

  - Polls the read replica until the latest `updated_at` timestamp is visible.

    ```
    SELECT updated_at FROM returns WHERE return_id = $1;
    ```

  - Once confirmed, updates the Redis cache with the current snapshot.

    ```
    SELECT return_id, status, eta_date, confidence, reason, history,
    updated_at FROM returns WHERE return_id = $1; SETEX return:{return_id} 300
    <serialized_json>
    ```

  - Emits a `status_cached` event for downstream services.

    ```
    { "return_id": "01J6WFCQFPJ5B0CBRHEPQK4AAG", "status": "APPROVED",
    "updated_at": "2025-02-06T15:45:00Z", "eta_date": "2025-03-20",
    "confidence": 0.94 }
    ```

- **AI Feed Worker**
  - Periodically runs refund ETA predictions using trained models.
  - Writes new `eta_date`, `confidence`, and `reason` back via the `/eta/predict` API.
  - Does not access the database directly; uses REST interface for isolation.
- **IRS Scrapper**
  - Periodically crawls
- **Status API Service**
  - Responsible for GET /status and Explain API endpoint

APIs

1️⃣ `GET /v1/status/{return_id}`

Purpose:

Retrieve the latest refund status and ETA for a specific tax return.

Request:

```
GET /v1/status/01HF6R8YZPCTZW3C84F6Z8Y5RT Authorization: Bearer <token>
```

Response:

```
{ "return_id": "01HF6R8YZPCTZW3C84F6Z8Y5RT", "status": "REFUND_APPROVED", "timeli
ne": [ {"stage": "FILED", "timestamp": "2025-04-01T13:00Z"}, {"stage": "ACCEPTED_
BY_IRS", "timestamp": "2025-04-02T09:00Z"}, {"stage": "REFUND_APPROVED", "timesta
mp": "2025-04-22T07:00Z"} ], "eta_date": "2025-05-12", "confidence": 0.86, "last_
updated": "2025-04-23T10:31:00Z" "etag": "W/\"f7a4f8e2\"", "request_id": "b231f7f
4e82d4b9c" }
```

Flow:

Status Service → Redis (Memorystore) → fallback Cloud SQL (read replica)

2️⃣ `POST /v1/status/explain`

Purpose:

Provide a clean interface to client apps where customer can interact with an LLM in the backend and get explanation on their refund status. Its a ChatGPT API stye Post + SSE Stream API.

Request

```
POST /status/explain Accept: text/event-stream Content-Type: application/json ---
------ Body ------------- { "return_id": "abc123", "session_id": "xyz789",
"question": "Why is my refund delayed?" }
```

Response (SSE Stream)

```
data: Your refund ETA is based on IRS processing times... data: Most delays this
year are due to staffing shortages... data: [DONE]
```

Sample Prompt to LLM

```
{ "system": "You are TurboTax's Refund Explanation Assistant. You explain refund
timelines clearly and empathetically using structured model context. Never reveal
internal model details, weights, or sensitive data.", "context": { "eta_date":
"2025-03-20", "confidence": 0.94, "confidence_band_days": [5, 9], "top_features":
[ {"feature": "refund_method", "value": "direct_deposit", "impact": -2.1},
{"feature": "tax_year", "value": "2024", "impact": +1.3}, {"feature":
"agi_bracket", "value": "100K-150K", "impact": +0.9}, {"feature": "filing_date",
"value": "2025-01-28", "impact": -0.7} ], "reason_summary": "Delays expected for
high-income filers this season.", "model_version": "v2.3.1", "inference_time":
"2025-02-06T15:45:00Z" }, "user_question": "Why is my refund taking longer this
year?", "conversation_history": [ {"role": "assistant", "content": "Your refund
is estimated to arrive by March 20 with high confidence (94%)."}, {"role":
"user", "content": "Why is it taking so long this year?"} ] }
```

Data Model

Returns

```
CREATE TABLE returns ( return_id CHAR(26) PRIMARY KEY, -- ULID per filing
filing_id CHAR(26) NOT NULL, -- Internal TurboTax reference user_id CHAR(26) NOT
NULL, -- Customer ID (or FK if needed) status VARCHAR(32) NOT NULL, -- e.g.
FILED, ACCEPTED, APPROVED, SENT, DEPOSITED eta_date DATE, -- Predicted refund
arrival date confidence NUMERIC(3,2), -- Prediction confidence (0.00–1.00)
last_source VARCHAR(32) DEFAULT 'IRS', -- Origin of the update (IRS / AI / MINT)
history JSONB DEFAULT '[]'::jsonb, -- Bounded array of ≤5 states (full timeline
for UI) updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL, -- Last update timestamp
updated_by VARCHAR(64), -- Service/component that made the change snap_context
JSONB, -- Optional structured data (IRS payload, model info) CONSTRAINT
valid_status CHECK ( status IN ('FILED','ACCEPTED','APPROVED','SENT','DEPOSITED')
) ); CREATE INDEX idx_returns_user_id ON returns(user_id); CREATE INDEX
idx_returns_updated_at ON returns(updated_at);
```

Sample Row

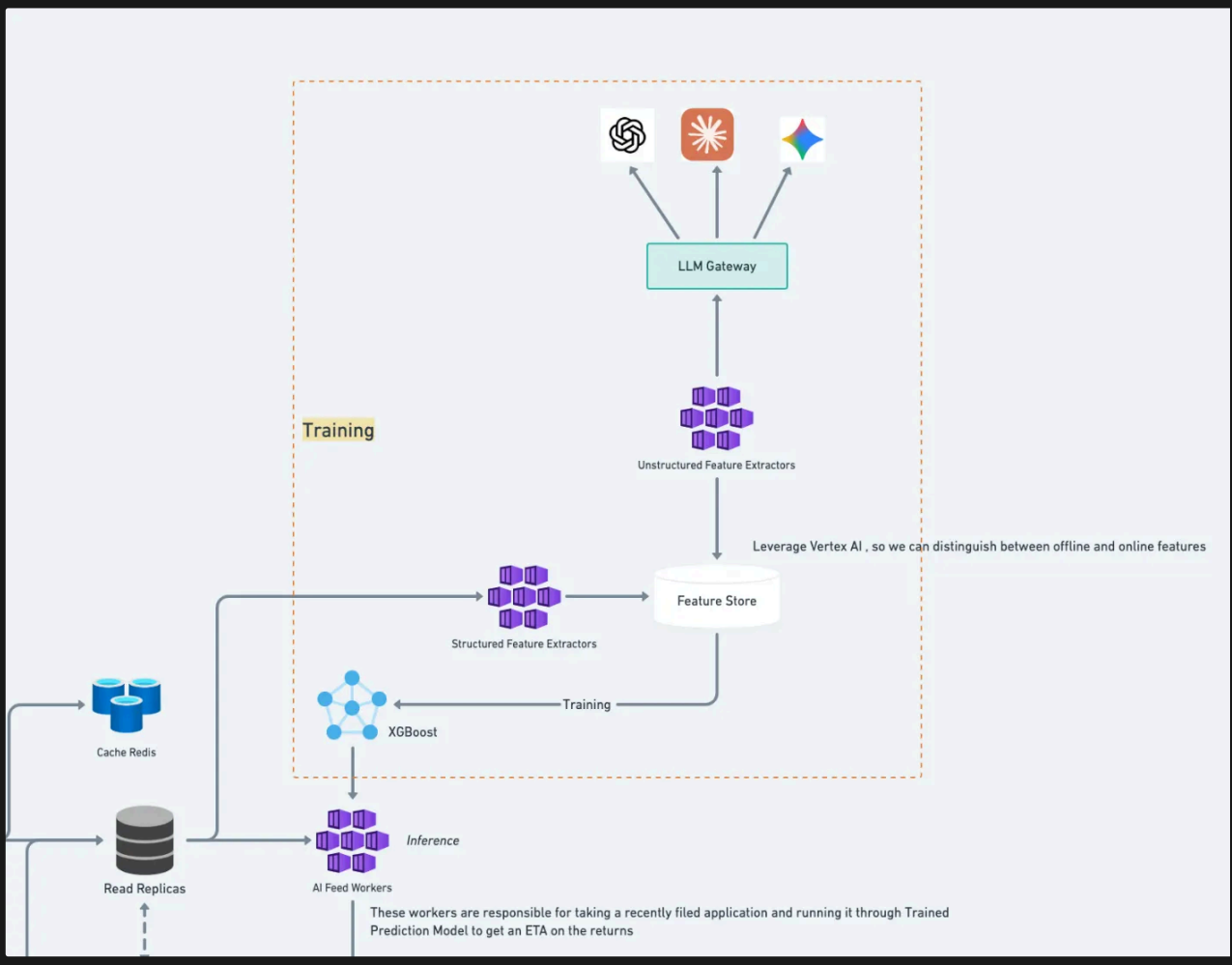
```
{ "return_id": "01J6WFCQFPJ5B0CBRHEPQK4AAG", "filing_id":
"01J6WFAFG8TT1YFNH6Y5M4H1W9", "user_id": "01J6WEPDTBHZ1P1C68CTAE45KG", "status":
"APPROVED", "eta_date": "2025-03-20", "confidence": 0.94, "source": "IRS",
"history": [ { "status": "FILED", "timestamp": "2025-02-01T08:00:00Z" }, {
"status": "ACCEPTED", "timestamp": "2025-02-03T09:12:00Z" }, { "status":
"APPROVED", "timestamp": "2025-02-06T15:45:00Z" } ], "updated_at": "2025-02-
06T15:45:00Z", "updated_by": "status_consumer", "snap_context": {
"model_version": "v2.3.1", "inference_time": "2025-02-06T15:45:00Z",
"top_features": [ { "feature": "refund_method", "value": "direct_deposit",
"impact": -2.1 }, { "feature": "tax_year", "value": "2024", "impact": +1.3 }, {
"feature": "agi_bracket", "value": "100K-150K", "impact": +0.9 }, { "feature":
"filing_date", "value": "2025-01-28", "impact": -0.7 } ], "reason_summary":
"Delays expected for high-income filers this season.", "confidence_band_days":
[5, 9], // ETA ± range "ai_worker_id": "vertex-ai-model-eta-23" } }
```

Filing

```
CREATE TABLE filings ( filing_id CHAR(26) PRIMARY KEY, -- ULID user_id CHAR(26)
NOT NULL, -- FK to TurboTax user return_id CHAR(26) UNIQUE, -- FK to
returns.return_id full_name TEXT NOT NULL, -- Taxpayer full name ssn_enc TEXT NOT
NULL, -- Encrypted digits of SSN filing_status VARCHAR(32) NOT NULL, -- Single /
Married / HOH refund_amount NUMERIC(10,2), -- Expected refund from return
filing_date DATE, -- When filed tax_year SMALLINT NOT NULL, -- e.g., 2024 agi
NUMERIC(12,2), -- Adjusted Gross Income income_bracket VARCHAR(16), -- e.g.,
“60K–80K” direct_deposit_routing VARCHAR(32), -- Bank routing number (masked or
encrypted) direct_deposit_account VARCHAR(32), -- Bank account (masked or
encrypted) refund_method VARCHAR(32), -- DirectDeposit / Check / Card created_at
TIMESTAMPTZ DEFAULT NOW(), updated_at TIMESTAMPTZ DEFAULT NOW() );
```

This is not a new table. This is an assumption that such table already exists.

Training Pipeline


LLM Gateway
Training
Unstructured Feature Extractors
Leverage Vertex AI , so we can distinguish between offline and online features
Feature Store
Structured Feature Extractors
Training
XGBoost
Cache Redis
Read Replicas
Inference
AI Feed Workers
These workers are responsible for taking a recently filed application and running it through Trained Prediction Model to get an ETA on the returns

🧱 1. Feature Engineering

- **Structured Extractors:** Build numeric/categorical features from `filings`, `irs_status`, and `mint_deposits` (e.g., refund amount, filing lag, income bracket).
- **LLM Extractors :** Use an LLM Gateway (OpenAI / Vertex / Anthropic) to derive latent features like *complexity_score*, *delay_risk*, and *missing_docs_flag*.
- **Feature Store:** Vertex AI Feature Store maintains parity between **offline (training)** and **online (inference)** features for reproducibility.

🧠 2. Model Training & Explainability

- **Modeling:** Train XGBoost regression models on combined structured + LLM features, optimized for MAE on `refund_lag_days`.
- **Explainability:** Generate SHAP-based feature importance artifacts for Explain API transparency ("Your refund is delayed due to high complexity").
- **Evaluation:** Validate with time-based splits, subgroup bias checks, and confidence calibration (isotonic/Platt scaling).

⚙️ 3. Orchestration & Governance

- **Pipeline Orchestration:** Vertex AI Pipelines handle DAG steps—feature build → train → eval → register → deploy—on nightly/weekly cadence.
- **Monitoring & Retraining:** Drift, bias, and calibration tracked continuously; retraining triggered automatically on drift or model version change.
- **Security & Versioning:** PII redacted via hashed SSNs; model lineage (data hash, transform version, LLM version) stored in `training_runs` and `model_registry`.

⚙️ 4. AI Feed Worker (Inference Worker)

- **Purpose:** A scheduled **cron job** that scans newly filed or recently updated returns (e.g., past 24 hours).
- **Operation:** For each return, it fetches precomputed features from the **Feature Store** and calls the deployed **ML model endpoint** (XGBoost in Vertex AI / Cloud Run) to get the predicted refund ETA and confidence.
- **Output:** Writes results back to the `returns` table (`eta_date`, `confidence`, `reason_summary`) and triggers downstream services such as the **Explain API** or user notifications.

Monitoring and Observability

Goal

Ensure the Refund ETA & Status Service maintains **correctness, freshness, and transparency** under production load.

Monitoring spans three layers — **application**, **infrastructure**, and **ML model performance** — with unified dashboards and automated alerting.

Service & Infrastructure Metrics

Status API & Cache Layer

- **QPS** — total and per-endpoint request volume.
  - *Target*: sustained 5K QPS with 95 % cache hit.

- **Latency** — p50 < 500 ms, p95 < 1 s.
- **Error Rate** — < 0.5 %.
- **Cache Hit Ratio** — should exceed 90 %. Drop below 80 % triggers investigation.
- **Cache TTL Expiry Rate** — ratio of expired keys refreshed vs. missed; expected < 5 %.
- **Redis Connection Utilization** — track `connected_clients` and latency via Cloud Monitoring.

Database Layer (Cloud SQL)

- **Replica Lag** — difference between master and read-replica `updated_at`.
  - *Target*: < 1 s; Warning > 3 s; Critical > 10 s.
- **Write Latency** — average transaction duration for `returns` updates < 100 ms.
- **Connection Pool Saturation** — < 80 %.
- **Deadlocks / Lock Waits** — watch `pg_stat_activity`.

Pub/Sub and Event Pipeline

- **Delivery Lag** — message publish → consume delay.
  - *Target*: < 1 s.
- **Retry Count / DLQ Volume** — alert if non-zero for `status_changes` or `status_cached`.
- **Replica Watcher Throughput** — # messages processed per minute; monitor drift vs. publish