

# Smarterouter Project — Full Report

2026-03-16

## Smarterouter Project Plan

Last Updated: 2026-03-16 Status: In Progress

---

### Overview

---

Field	Details
<b>Project Name</b>	Smarterouter Project
<b>Start Date</b>	2026-02-18
<b>Target Completion</b>	TBD
<b>Owner</b>	TBD
<b>Status</b>	Planning

---

### Goals & Success Criteria

#### Purpose

This is a **scoping and assessment project**. The goal of this phase is to nail down everything that needs to be done and produce a clear, detailed estimate of the effort required to integrate Athia AI/ML into Deuna's payments service. No implementation is happening yet.

#### Phase 0 Deliverables (Current Focus)

- Full understanding of Deuna's data, schema, and existing routing infrastructure
- Detailed breakdown of all work required across P-01 through P-05 use cases
- Effort estimates per workstream (engineering, data, infra, ML)
- Risk identification and open questions resolved
- Clear recommendation on what to build and in what order

#### Success Criteria (Phase 0)

- All open questions answered
- Effort estimate produced with confidence
- All access and dependencies identified and documented
- Stakeholder alignment on scope, timeline, and approach

#### Longer-Term Success Criteria (Post-Scoping)

- Measurable approval lift
- Stability during PSP outages
- Latency target: p95 < 200ms

## In Scope (Phase 0 — Assessment Only)

- Understand Deuna’s data, schema, and existing routing rules
- Assess Athia platform gaps vs. what’s needed
- Size effort for: Processor/Message selector, Smart Retry logic, Feedback Loop
- Identify all dependencies, blockers, and risks

## Out of Scope (Phase 0)

- Any implementation or code delivery
- 3DS optimization (Phase 2)
- User-facing messaging (Phase 3)
- Installment optimization

---

## Stakeholders

See TEAMS.md for the full source of truth on all people and roles.

---

Name	Company	Role
Pablo	Deuna	CTO
Israel	Deuna	Data POC
Farhan	Deuna	Claude/LLM Access POC
Mark Walick	Deuna	PM Lead
Rakesh	Aidaptive	Engineer
Naoki	Aidaptive	Engineer

---

## Milestones & Timeline

---

Milestone	Duration	Status	Notes
Phase 0: Assess level of effort/complexity	2 days	In Progress	\$6K budget, started 2026-02-18
Phase 1: Model running in production for 2 processors with basic feature store	2 weeks	Pending	Core delivery
Phase 2: Add monitoring + integrate with experimentation	Week 3	Pending	Immediately after Phase 1
Phase 3: Drift detection, CI/CD, experiment ramp-up, additional model techniques	TBD	Pending	

---

---

## Key Use Cases (P0)

---

ID	Use Case	Description
P-01	Outage detection & failover	Fail over/back via persistent timeout codes; random sampling of down PSP to detect recovery

---

ID	Use Case	Description
P-02	Overall transaction routing optimizer	Optimize Deuna's existing static rules based on historical outcomes
P-03	Per-transaction optimal route selection	Rank top 3 routes based on prior outcomes for fast retry
P-04	Message manipulation	Toggle CIT/MIT, AVS, MCC variables in authorization requests; top 3 recommendations
P-05	Retry optimization	Subs/MIT focused; enterprise darktime reduction; delayed retry based on reputation

---

## Work Breakdown / Task Tracking

### Backlog

- Claude access and LLM budget provisioned (2026-02-19)
- Confirm Snowflake read access provisioned — Rakesh verified (2026-02-18, info from Israel)
- Naoki Snowflake access confirmed (2026-02-19)
- Provision Deuna corp accounts for Rakesh and Naoki
  - Snowflake instance access for both accounts
  - Code (repo) access for Rakesh — granted by Pablo (2026-02-19)
  - Code (repo) access for Naoki
    - [~] Claude Code credits for both accounts — Not needed
- Complete Phase 0: assess level of effort/complexity (2 days, \$6K)
- Build training platform (currently prototype-only — see Technical Gaps)
- Deliver P-01 through P-05 use cases

### In Progress

- (nothing yet)

### Done

- (nothing yet)
- 

## Schema Understanding & Data Notes

Extracted 2026-02-18 from PAYMENT\_ML Snowflake database. Full schema reference: SCHEMA.md

### Overall Assessment

The schema is very well structured for the P0 use cases. The data is organized into clean source views in the **SOURCES** schema, and a massive denormalized flat table (**ABTESTING.ALL\_VIEWS\_FLAT**) that joins everything together — ideal for quick EDA and feature engineering without complex joins.

---

### Key Tables for P0 Use Cases

**VW\_ATHENA\_PAYMENT\_ATTEMPT** — most important table for routing & retry - Tracks every individual attempt with sequence order, processor used, error code/category, hard/soft decline type, retry indicator, and approved status - **DYNAMIC\_ROUTING\_DETAIL** (**VARIANT/JSON**) column likely contains rich routing decision metadata — needs exploration - **PAYMENT\_ATTEMPT\_SEQUENCE\_ORDER** + **PAYMENT\_LAST\_ATTEMPT\_INDICATOR** make it easy to reconstruct the full retry chain per payment - Directly supports **P-03** (per-transaction route selection) and **P-05** (retry optimization)

**VW\_SMART\_ROUTING\_ATTEMPTS** — current routing engine event log - Captures per-attempt routing decisions: algorithm type, processor selected, process time, result status, skip reason - **PROPERTIES\_RESULT\_PROCESS\_TIME** is a direct latency signal for the p95 < 200ms target - **PROPERTIES\_RESULT\_SKIPPED\_REASON** tells us why processors were bypassed — key for **P-01** (outage detection) - **PROPERTIES\_ALGORITHM\_TYPE** reveals what routing strategies are already in use

**VW\_ROUTING\_MERCHANT\_RULE + related views** — existing rules engine - Deuna already has a rules-based routing system with conditions, members, options, and priority ordering - This is the foundation for **P-02** (optimize existing static rules) — we don't start from scratch - **SHADOW\_MODE** in **VW\_ROUTING\_MERCHANT\_RULE\_MEMBER** suggests there's already infrastructure for testing new processors without live traffic

## Feature Richness for ML Models

The data has strong signal across multiple dimensions:

Feature Group	Key Columns	Usefulness
Retry history	NUM_ATTEMPTS_ORDER, PREVIOUS_ORDER_ERROR_CODE, PREVIOUS_ORDER_PROCESSOR, AVG_SEC_BETWEEN_PAYMENT_ATTEMPS	Direct retry optimization signals
Error signals	ERROR_CODE, ERROR_CATEGORY, HARD_SOFT, EVENT_ERROR_STANDARD_ERROR_CODE	Distinguish hard vs soft declines; normalized error codes in events
Card signals	CARD_BIN, CARD_BRAND, BANK, CARD_COUNTRY	Processor affinity by card type
User behavior	TARGET_USER_FRAUD_RATE_COHORT, TARGET_USER_TENURE_IN_DAYS, TARGET_USER_FREQUENCY_VALUE, TOTA_MINUTES_BROWSING, TOTAL_NUM_SESSIONS	User risk and engagement signals
RFM	TARGET_USER_FREQUENCY_VALUE, TARGET_USER_RECENCY_VALUE, TARGET_USER_MONETARY_VALUE	Customer value for routing priority
Geo	LATITUDE, LONGITUDE, ORDER_COUNTRY_CODE, WEATHER_MAIN	Geography-based processor routing
Device	TARGET_USER_BROWSER, TARGET_USER_OS, TARGET_USER_DEVICE	Device fingerprinting
Message config	MCI_MSI_TYPE, ORDER_MCI_MSI_TYPE, PAYMENT_ATTEMPT_METHOD_TYPE	CIT/MIT toggle tracking for <b>P-04</b>
3DS	CHALLENGE_3DS_INDICATOR, CHALLENGE_3DS_STATUS	Available now; scoped to Phase 2

## Starting Point Recommendation

- Use **ABTESTING.ALL\_VIEWS\_FLAT** for initial EDA — everything is already joined
- Switch to individual **SOURCES** views for model training to avoid data leakage and redundancy
- Explore **DYNAMIC\_ROUTING\_DETAIL VARIANT** column in **VW\_ATHENA\_PAYMENT\_ATTEMPT** early — may contain routing features not exposed elsewhere

## Notable Data Quality Observations

- **Typo in source data:** `PATMENT_TIME` in `VW_ATHENA_PAYMENT` (should be `PAYMENT_TIME`) — minor but worth noting for pipelines
- **Airline-specific data:** `VW_ORDER_AIRLINE_DETAIL_ALL` and `VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL` suggest Volaris is a key merchant with rich flight/passenger metadata
- `SOURCES` schema has no raw tables — only views, meaning underlying raw tables are managed upstream by Deuna's data team (Israel's domain)

---

## Technical Gaps (from SOW)

### Testing/Experimentation Platform — Production Ready

- A/B testing infrastructure, multi-variant experiments, automated winner selection
- Model registry with versioning, real-time inference (FastAPI sidecar)
- Missing: canary deployments

### Training Platform — Prototype Only (needs significant work)

#### Current State

Component	Status	Notes
Basic training scripts	Prototype	3 models — NOT production-ready
Snowflake data access	Partial	Manual queries only
Logistic Regression model	Prototype	No real ML pipeline

## Gap Tracking

#	Gap	Category	Priority	Status	Notes
G-01	Automated retraining	Automation	High	<b>Done</b>	<code>training_pipeline.py</code> + <code>llm_training_orchestrator.py</code> (feat/ATH-0000)
G-02	Orchestration	Infrastructure	High	<b>Partial</b>	<code>schema_refresh_scheduler.py</code> handles daily refresh; no full DAG orchestration yet
G-03	CI/CD pipeline	DevOps	High	<b>Partial</b>	CI improvements: black, isort, uv, pytest fixtures. No full CD pipeline
G-04	Data validation	Data Quality	High	<b>Done</b>	<code>data_quality_validator.py</code> (834 lines): temporal, balance, outlier, drift (feat/ATH-0000)
G-05	Model monitoring	Observability	High	<b>Nearly Done</b>	OTEL metrics pipeline + Grafana CloudWatch dashboard (v0.15.1). Missing: alerting thresholds

#	Gap	Category	Priority	Status	Notes
G-06	Deployment automation	Automation	High	<b>Nearly Done</b>	Triton IS + <code>ModelConversionManager</code> + <code>ServiceWithTriton</code> define complete path; ~1.5d wiring left (athena-platform Triton branch)
G-07	Model registration automation	Automation	Medium	<b>Done</b>	<code>model_registry.py</code> auto-creates tables; new <code>ExperimentService</code> one-call API (Triton branch)
G-08	Feature store	ML Infrastructure	High	<b>Partial</b>	<code>feature_extractor.py</code> + v2 encoder with feature mappings on main. Feature Service on feature branch
G-09	Drift detection	Observability	Medium	<b>Done</b>	<code>data_quality_validator.py</code> : feature drift, prediction drift (KS + PSI), concept drift (feat/ATH-0000)
G-10	Lineage tracking	Governance	Medium	Not started	Explicitly noted as missing; <code>TrainingDatasetVersion</code> not implemented
G-11	Hyperparameter tuning	ML Quality	Medium	<b>Done</b>	<code>llm_experiment_designer.py</code> : grid/random search per algorithm; trials logged to Snowflake (feat/ATH-0000)
G-12	Algorithm comparison	ML Quality	Medium	<b>Done</b>	<code>training_pipeline.py</code> supports LR, RandomForest, XGBoost; champion auto-selected by F1 (feat/ATH-0000)
G-13	Versioning workflow	Governance	High	<b>Nearly Done</b>	Auto versioning + dynamic per-model encoders + A/B version routing on main. Missing: retention policies

#	Gap	Category	Priority	Status	Notes
G-14	Rollback capability	Reliability	High	<b>Partial</b>	Shadow mode ( <code>is_shadow_mode</code> ) + <code>is_default</code> fallback = safe rollback mechanism (Triton branch); one-command API not yet built

### Summary by Category

Category	Gaps	Notes
Automation	G-01, G-03, G-06, G-07	Core pipeline work — needed before any production use
Infrastructure	G-02, G-08	Orchestration + feature store are foundational
Observability	G-05, G-09	Monitoring + drift detection — needed post-deploy
Governance	G-10, G-13	Lineage + versioning — important for auditability
Reliability	G-14	Rollback — critical for safe production deployment
ML Quality	G-11, G-12	Nice to have in Phase 1; important for long-term quality
Data Quality	G-04	Validate inputs before training

### Effort Assessment

To be filled in during Phase 0 assessment.

Gap	Original Estimate	Saved	Remaining	Dependencies	Owner
G-01 Automated retraining	10d	<b>10d</b>	0d	G-02, G-08	TBD
G-02 Orchestration	8d	~2d	~6d		TBD
G-03 CI/CD	9d	~1d	~ <b>8d</b>		TBD
G-04 Data validation	7d	<b>7d</b>	0d	G-08	TBD
G-05 Model monitoring	8d	~5d	~3d	G-06	TBD
G-06 Deployment automation	7.5d	~6d	~1.5d	G-03	TBD
G-07 Model registration automation	5.5d	<b>5.5d</b>	0d	G-06	TBD
G-08 Feature store	13.5d	~5d	~8.5d		TBD
G-09 Drift detection	7d	<b>7d</b>	0d	G-05	TBD
G-10 Lineage tracking	6.5d	0	~ <b>6.5d</b>	G-07, G-13	TBD
G-11 Hyperparameter tuning	5.5d	<b>5.5d</b>	0d	G-01	TBD
G-12 Algorithm comparison	7d	<b>7d</b>	0d	G-01	TBD
G-13 Versioning workflow	5d	~4.5d	~0.5d	G-06	TBD
G-14 Rollback capability	5d	~4d	~1d	G-13	TBD
<b>Total</b>	<b>~104.5d</b>	<b>~75.5d</b>	<b>~29d</b>		

### Recommended Build Order

1. **Foundation:** G-02 Orchestration → G-08 Feature store → G-04 Data validation
2. **Automation:** G-03 CI/CD → G-06 Deployment automation → G-07 Model registration → G-01 Automated retraining
3. **Governance:** G-13 Versioning → G-10 Lineage → G-14 Rollback

4. **Observability:** G-05 Monitoring → G-09 Drift detection
  5. **ML Quality:** G-11 Hyperparameter tuning → G-12 Algorithm comparison
- 

## Next Steps

### Immediate Blockers to Resolve

- Claude/LLM access & budget — provisioned (2026-02-19)
- Provision Deuna corp accounts for Rakesh & Naoki (Snowflake, code/repo, Claude Code credits)
- Naoki Snowflake access confirmed (2026-02-19)

### Phase 0 Assessment Work

- Get data export or pointers to which Snowflake table(s) contain all data needed to train Volaris smart routing model
- Explore DYNAMIC\_ROUTING\_DETAIL JSON column in VW\_ATHENA\_PAYMENT\_ATTEMPT — likely contains rich routing metadata not exposed elsewhere
- Run EDA on ABTESTING.ALL\_VIEWS\_FLAT — understand data volumes, date ranges, merchant mix, approval rates, processor distribution
- Map existing routing rules — query VW\_ROUTING\_MERCHANT\_RULE and related views to understand current rule engine
- Assess Athia training platform gaps — produce concrete list of what needs to be built vs. what already exists
- Size effort per use case (P-01 through P-05) — engineering, data, infra, and ML effort per workstream
- Identify risks and open questions — populate the Risks and Open Questions sections below

### Documentation & Alignment

- Fill in Open Questions section — capture anything still unclear from the Deuna side
  - Produce final Phase 0 deliverable — effort estimate doc with work breakdown, risks, and recommended build order for stakeholder sign-off
- 

## Decisions Log

Date	Decision	Rationale	Made By
2026-02-18	Latency target updated from p95 < 50ms to p95 < 200ms	Revised from original SOW spec	Rakesh (discussed with Pablo)
2026-02-19	Target merchant for Phase 1 set to <b>Volaris</b> (not Cinépolis)	Volaris has known PSPs (Worldpay ID:76, MIT ID:85, Elavon, Amex); Cinépolis only shows Cybersource gateway with unknown processor behind it	Mark Walick
2026-02-20	Repo analysis scoped to branch <code>feat/ATH-0000-athia-ml-llm-scraper-discovery</code> (not main)	This branch contains the active ML platform <del>scope discovery</del> does not reflect current capabilities	Pablo
2026-03-13	Adopt <b>TensorFlow ecosystem</b> (TFX, TF Serving, TFDV, TFMA, TF Transform) for all ML work	Replaces Snowflake ML / XGBoost / scikit-learn. Provides unified training → validation → serving pipeline with production-grade tooling	Rakesh

Date	Decision	Rationale	Made By
2026-03-13	Implement <b>7 service shells</b> before Volaris feature work — defines service boundaries for all ML infrastructure	Data Pipelines, Feature Service, Training Pipelines, Model Mgmt, Eval Service, Evaluation Framework, Experiment System	Rakesh
2026-02-26	athena-platform <b>feature/llm-driven-ml-training</b> branch identified as production model serving path	Triton IS + shadow mode + <b>ExperimentService</b> provides complete training→serving pipeline; replaces manual SageMaker endpoint registration	Rakesh

## Open Questions

#	Question	Owner	Status
1	Claude/LLM access & budget — when will this be provisioned?	Pablo → Farhan	<b>Done</b> (2026-02-19)
2	Are ATHIA_PREDICTIONS / ATHIA_FEEDBACK tables populated in Deuna's Snowflake today?	Israel / Rakesh	<b>Confirmed</b> (2026-02-24)
3	Are SageMaker endpoints currently live for processor_selector / retry_predictor?	Rakesh	Open
4	Is there a live model in MODEL_ARTIFACTS that Deuna's payment service is calling today?	Rakesh	Open
5	What is the current payment volume through the routing engine? (Validates A/B test sample size feasibility)	Israel	Open
6	Who owns athena-platform Go repo deployments — Aidaptive or Deuna infra?	Pablo / Rakesh	Open

## Risks & Issues

ID	Description	Likelihood	Impact	Mitigation	Status
R1	TBD	Low/Med/High	Low/Med/High	TBD	Open

## Repository Analysis & Code Intelligence

**Analyzed:** 2026-02-19 — Both Deuna repos cloned and analyzed in full. **Repos:** DATA-Athena-Snowflake | athena-platform

---

## Repo 1: DATA-Athena-Snowflake — LLM Analytics + ML Training Platform

**Branch:** `main` (`feat/ATH-0000-athia-ml-llm-schema-discovery` merged to `main`, 182 files, +29K lines) **13 ML services now on main.** Training pipeline at `src/services/training_pipeline.py`.

**What it is:** A Python-based (FastAPI + LangGraph/LangChain) multi-agent AI platform with a complete ML training pipeline. This is Athia's data intelligence layer — it uses LLMs (GPT-4, Claude via Anthropic/Bedrock) to analyze Snowflake data, detect anomalies, generate payment optimization strategies, and autonomously design and execute ML training experiments.

**ML training is now on main.** All 13 services (TrainingPipeline, LLMTrainingOrchestrator, ExperimentDesigner, ModelDeployer, FeatureExtractor, DataQualityValidator, FeedbackCollector, ModelRegistry, SchemaDiscovery, TrainingPlanner, SchemaRefreshScheduler, and more) are available on the main branch.

### Architecture

- **Framework:** FastAPI + LangGraph (stimulus-response multi-agent pattern)
- **LLM Backend:** Anthropic Claude (primary), OpenAI GPT-4 (fallback) — via `LLMProvider` abstraction
- **Data Layer:** Snowflake Snowpark with async session pooling, Jinja2-templated SQL queries
- **ML Backend:** Snowflake ML (`snowflake.ml.modeling`) — LogisticRegression, RandomForest, XGBoost
- **Vector Store:** ChromaDB (embedded) — schemas, past experiments, training decisions stored for RAG
- **Deployment:** AWS Lambda + ECS + Bedrock AgentCore + SQS, CI/CD via GitHub Actions

### Implemented Workflows (11 stimuli, unchanged from main)

Stimulus	Purpose	Status
<code>acceptance_rate_analysis_requested</code>	Detect acceptance rate drops, processor issues, BIN anomalies	Implemented ( <code>v0_1</code> , <b><code>v1_0</code> new</b> )
<code>fraud_card_analysis_requested</code>	Fraud detection: card testing, false positives, geographic patterns	Implemented ( <code>v0_1</code> )
<code>cost_optimization_analysis_requested</code>	Payment processing cost analysis	Early stage
<code>strategy_generation_initiated</code>	Orchestrate analysts and rank strategic recommendations	Partially implemented
<code>metrics_anomaly_research_triggered</code>	Automated anomaly detection and trend analysis	Implemented
<code>user_question_submitted</code>	Conversational chatbot for data queries	Implemented
<code>data_analyst_requested</code>	SQL generation and data visualization	Implemented
<code>researcher_assistance_requested</code>	Comprehensive research with parallel deep-dive explorers	Implemented
<code>deep_exploration_needed</code>	Root cause analysis for metric anomalies	Implemented
<code>element_edition_requested</code>	SQL query modification	Implemented
<code>knowledge_expert_asked</code>	External knowledge base via MCP	Implemented

**ML Training Platform (now on main)** 15 new services, 7 new route files, 6 new test files, new SQL tables.

**New Services:** | Service | Lines | Purpose | |—|—|—| | `training_pipeline.py` | 777 | Executes real Snowflake ML training (LR, RF, XGBoost); test-mode fallback | | `training_planner.py` | 724 | Dry-run planning (like `terraform plan`) — no execution | | `model_registry.py` | 608 | Model lifecycle; auto-creates `{MODEL}_PREDICTIONS`, `_FEEDBACK`, `_TRAINING_DATASET` | | `schema_discovery.py` | 421 | Auto-discovers Snowflake tables; stores in ChromaDB with embeddings | | `feature_extractor.py` | 657 | Feature engineering from historical tables or `ATHIA_PREDICTIONS` | | `data_quality_validator.py` | 834 | Temporal bias, class balance, outlier, concept drift validation | | `feedback_collector.py` | 569 | Feedback collection in 3 modes: webhook, API poll, batch reconcile | | `llm_experiment_designer.py` | 563 | GPT-4 + RAG → designs 7 diverse experiments (simple→complex) | |

llm\_training\_orchestrator.py | 560 | GPT-4 + RAG → autonomous retraining decisions (RETRAIN\_NOW / SCHEDULED / SKIP) | | model\_deployer.py | 432 | EFS export + athena-platform payload prep (API integration incomplete) | | schema\_refresh\_scheduler.py | 121 | APScheduler daily schema refresh at 2 AM UTC | | schema\_understanding.py | 354 | LLM-powered schema analysis: identifies labels vs. features | | warehouse\_selector.py | 88 | Warehouse management from env var / override | | athia\_ingestion.py | 584 | Event ingestion into ATHIA\_PREDICTIONS / ATHIA\_FEEDBACK / SESSION / STAGE tables | | llm\_provider.py | 232 | Anthropic Claude primary + OpenAI GPT-4 fallback |

**New API Endpoints:** | Route | Key Endpoints | |—|—| | /api/v1/training/ | POST /plan/{model\_type}, POST /run/{model\_type}, POST /deploy/{model\_type}, GET /history/{model\_type} | | /api/v1/training/decision/ | POST /{model\_type} (LLM orchestrator decision), GET /decisions (all models) | | /api/v1/models/ | Full CRUD: register, list, get, update, deactivate | | /api/v1/schemas/ | GET /, GET /search, GET /{schema}/{table}, POST /refresh | | /api/v1/ingest/athia-events | Ingest prediction + feedback events | | /webhooks/{model\_type}/feedback | Webhook feedback receiver | | /api/v1/feedback/ | poll, reconcile, coverage, alerts | | /api/v1/experiments/ | POST /design/{model\_type}, GET /recommendations/{model\_type} |

**New SQL (this branch deploys these):** | Table/View | Notes | |—|—| | ML\_MODEL\_REGISTRY | Model config registry with cost limits, training frequency, feedback config | | ATHIA\_PREDICTIONS | Fully defined with 50+ columns: card, session, geo, fraud, request/response payloads | | ATHIA\_FEEDBACK | Fully defined: per-stage outcomes (installation, processor, retry) | | ATHIA\_SESSION\_SUMMARY | **Now deployed** (was “designed, not deployed” on main) | | ATHIA\_STAGE\_OUTCOMES | **Now deployed** (was “designed, not deployed” on main) | | ATHIA\_TRAINING\_DATASET | View joining PREDICTIONS + FEEDBACK on PREDICTION\_ID | | ATHIA\_EXPERIMENT\_LIFT | Statistical significance (z-score, p-value) by experiment group | | ATHIA\_MULTI\_STAGE\_ANALYSIS | Multi-stage funnel: installation → processor → retry | | ATHIA\_MODEL\_METRICS | Daily accuracy, latency p50/p95/p99 per model version |

#### End-to-end training flow:

```
POST /training/run/processor_selector
  → Schema Discovery (ChromaDB index)
  → Data Quality Validation (temporal, balance, drift)
  → LLM Training Orchestrator (RETRAIN_NOW / SKIP)
  → LLM Experiment Designer (7 experiments, simple→complex)
  → Snowflake ML Training (LR, RF, XGBoost; 80/20 temporal split)
  → Best model selected by F1
  → Export to EFS + athena-platform payload prepared
  → Results stored in ML_TRAINING_RUNS
```

**Production readiness: ~85%.** Core training solid. Deployment to athena-platform (API registration, canary creation) not yet automated.

#### Relevance to P0 Use Cases

- **P-02 (Routing optimizer):** acceptance\_rate\_analysis\_requested v1\_0 upgraded with richer tools and data enricher. Still LLM-based recommendations, not trained model.
- **P-03 (Per-transaction route selection):** processor\_selector model type fully supported by training pipeline.
- **P-05 (Retry optimization):** retry\_predictor and retry\_sequence model types supported. No dedicated LLM workflow yet.
- **P-01 (Outage detection):** Still lives in athena-platform serving layer, not here.

#### Remaining Gaps in This Repo (post-branch)

- **Strategy Director still incomplete:** Matcher exit() placeholder + dummy Ranker prompts unchanged.
- **No retry LLM workflow:** retry\_optimization\_requested stimulus still missing (P-05 gap).
- **Deployment integration incomplete:** Model deployer exports to EFS but does NOT call athena-platform API — manual steps still required.
- **No data lineage tracking:** TrainingDatasetVersion not implemented — can't trace which data produced which model (G-10).
- **No rollback capability:** G-14 not addressed.

- **SQL injection risks:** String interpolation in `schema_discovery.py`, `training_planner.py`, `athia_ingestion.py`, and `acceptance_rate v1_0` branch.py.
- **Fragile LLM JSON parsing:** All services extract JSON by searching for braces — not robust.
- **No circuit breaker:** LangGraph node failures still cascade.

## Repo 2: athena-platform — ML Serving & Experimentation Platform

**What it is:** A production Go (Gin) REST API that serves ML predictions, manages model experiments (A/B testing), and handles the full model lifecycle. This is the component that Deuna’s payment service calls to get routing decisions in real time.

### Architecture

- **Language/Framework:** Go + Gin (Clean Architecture)
- **Databases:** PostgreSQL (RDS Multi-AZ) + Redis (ElastiCache) for caching
- **ML Backends:** AWS SageMaker, Snowflake Cortex, custom HTTP endpoints
- **Snowflake Integration:** JWT-based private key auth; queries `ATHIA_PREDICTIONS`, `ATHIA_FEEDBACK`, `ATHIA_TRAINING_DATASET`, `ATHIA_EXPERIMENT_LIFT`
- **Deployment:** ECS Fargate + ALB (mTLS enforced for `/api/v1/ml/predict/*`) + EFS for model storage

### ML Model Registry (Already Implemented)

Inference Type	Description	Maps to Use Case
<code>processor_selector</code>	Ranks available processors by approval probability	P-03: Per-transaction route selection
<code>retry_predictor</code>	Predicts retry success probability	P-05: Retry optimization
<code>retry_sequence</code>	Predicts optimal retry processor order	P-05: Retry optimization
<code>installment_optimizer</code>	Predicts best installment options	Out of scope Phase 1

**Key finding:** The model registry schema (`MODEL_ARTIFACTS`, `MODEL_EXPERIMENTS`, `MODEL_EXPERIMENT_VARIANTS`) and inference API are already built. **What’s missing is the training pipeline** that produces models to register here.

### A/B Experimentation System (Production-Ready)

- **Bucketing:** `SHA256(transaction_id) % 10000` — deterministic and reproducible
- **Traffic splits:** Basis points (bps) — e.g., 30% control / 70% treatment
- **Auto-winner evaluation:** Statistical significance testing with full guardrails:
  - Minimum 7 days runtime
  - Minimum 1000 samples per variant
  - p-value < 0.05 (95% confidence)
  - Minimum 1% absolute lift
  - Latency regression guard: 10%
  - Revenue regression guard: -5%
- **Dry run mode:** Safe default for testing evaluation logic before enabling real rollouts
- **Merchant-specific experiments:** Experiments can be scoped to specific merchant IDs
- **Shadow mode** (merged to main v0.15.0): `is_shadow_mode=true` + `is_default` fallback seed experiments for all 3 model types — safe production testing without affecting routing
- **Max variants** (merged to main): Configurable per experiment (default: 3); variant deprecation by ID when replacing
- **OTEL metrics** (v0.15.1): Full OpenTelemetry metrics wired for model inference pipeline + Grafana Cloud-Watch dashboard
- **Dynamic per-model encoders** (v0.15.2+): V2 processor selector encoder with updated feature derivation, new error stats, refined interaction ARs. A/B model version routing.

- **Prediction event handling** (ATH-1140): Flexible metadata support for prediction events
- **Version: v0.15.5** (released 2026-03-11) — includes all Triton, shadow mode, OTEL, and v2 encoder improvements

## Snowflake Tables for Training & Monitoring

Table	Purpose	Status
ATHIA_PREDICTIONS	Model inputs + outputs per prediction	Active
ATHIA_FEEDBACK	Transaction outcomes (approved/declined)	Active
ATHIA_TRAINING_DATASET	Predictions + feedback joined for retraining	Active
ATHIA_EXPERIMENT_LIFT	Aggregated experiment metrics for auto-winner	Active
ATHIA_STAGE_OUTCOMES	Per-stage funnel outcomes (installment → processor → retry)	<b>Designed, not deployed</b>
ATHIA_SESSION_SUMMARY	Full session-level aggregations	<b>Designed, not deployed</b>

## Gaps Found in This Repo (Updated 2026-02-26)

- **Analytic tables not deployed** — ATHIA\_STAGE\_OUTCOMES and ATHIA\_SESSION\_SUMMARY now deployed in feat/ATH-0000 branch.
- **No production monitoring dashboards:** Grafana setup exists locally (docker-compose) but no alert rules or production dashboards are configured.
- **Training pipeline integration:** Triton branch (feature/llm-driven-ml-training, in review) provides complete model serving path. Training→Triton wiring in model\_deployer.py needs ~1.5d to complete.
- **Cache invalidation is manual:** Experiment assignment cache (Redis, 24h TTL) has no auto-invalidation on config changes.
- **LLM/Bedrock integration:** AgentCore VPC endpoint now supported (v0.14.0); full agent orchestration still in progress.
- **No rate limiting:** No global rate limiting middleware — only per-user, per-feature quotas.
- **V2 API (18 handlers) completely untested** — entire new API version still at 0% coverage.

## Impact on Effort Estimates

This analysis materially changes our understanding of the effort required. Here is what we learned:

## What's Already Built (Reduces Effort)

Area	What Exists	Gap Gaps Affected
Model serving API	Full Go REST API with processor_selector + retry_predictor endpoints	Reduces P-03, P-05 integration work
Model registry	ModelArtifact + Experiment + ExperimentVariant tables + CRUD API	Reduces G-07 effort
A/B experimentation	Full auto-winner system with statistical guardrails	Reduces P-02 work
Snowflake feedback loop	ATHIA_PREDICTIONS + ATHIA_FEEDBACK + ATHIA_TRAINING_DATASET tables	Reduces G-05, G-08 baseline work
LLM analytics	Acceptance rate + fraud workflows for insights	Can accelerate P-02 analysis

Area	What Exists	Gap Gaps Affected
CI/CD skeleton	GitHub Actions workflows for both repos	Reduces G-03 baseline work

### What Still Needs to Be Built

Area	Specific Work	Gaps
Training pipeline	Automated end-to-end training → model registration	G-01, G-06, G-07
Feature store	Consistent feature computation + serving (Redis-backed)	G-08
Orchestration	Scheduled + trigger-based training runs	G-02
Analytics tables	Deploy ATHIA_STAGE_OUTCOMES + ATHIA_SESSION_SUMMARY in Snowflake	G-05
Production monitoring	Grafana dashboards + alerting rules for model performance	G-05, G-09
Retry workflow (LLM)	New stimulus <code>retry_optimization_requested</code> in DATA-Athena-Snowflake	P-05
Strategy Director	Finish matcher + ranker nodes in DATA-Athena-Snowflake	P-02
Data validation	Add schema + statistical validation to training pipeline	G-04
Versioning	Automated version bumping + metadata tagging on training runs	G-13
Lineage tracking	Record data → feature → model → deployment lineage	G-10
Rollback capability	One-command rollback to previous registered model	G-14
Drift detection	Feature + prediction distribution monitoring	G-09
Hyperparameter tuning	Integrate Optuna/Ray Tune into training pipeline	G-11
Algorithm comparison	Add XGBoost/LightGBM/NN alternatives to Logistic Regression	G-12

## Testing Coverage & Architecture Deep-Dive

**Analyzed:** 2026-02-19 — Both repos examined in full for testing maturity and architectural patterns.

### DATA-Athena-Snowflake — Testing Coverage

**Branch analyzed:** feat/ATH-0000-athia-ml-llm-schema-discovery (2026-02-20)

Metric	Detail
Test files	28 (existing) + 6 new root-level integration tests

Metric	Detail
Test functions	421 (existing) + ~30 new
Test frameworks	pytest 8.4, pytest-asyncio, pytest-mock + new integration tests (no mocking)
CI	Separate workflows for metrics + deployment only — no unified suite
Estimated coverage	~ <b>20%</b> (up from 18%; new tests are integration-heavy)

**Well tested:** - Metrics layer: 70–80% (5 test files, strong validation tests) - Deployment config/validation: 50–60% - Deep-dive explorer utilities (pareto filter, metric router, data comparator) - New: model registry CRUD (~60% happy-path coverage) - New: schema discovery + semantic search (~45%) - New: experiment designer + RAG (~40%) - New: training orchestrator + RAG (~35%) - New: feedback collector (~35%)

**Critical gaps (untested):** - All route handlers — still largely 0% - All 3 clients (Redis, Postgres, Platform) — 0% - Core multi-agent framework (**AgentWorkflow**, **AgentStrategy**, **ToolStrategy**) — still not in pytest - All 11 workflow branches — no unit tests - All 4 Lambda / AgentCore entrypoints — 0% - Middleware stack — 0% - New ML services: error handling, edge cases, external dep failures all untested - All 6 new test files depend on live Snowflake + OpenAI (not mocked — not repeatable in CI)

**Maturity: 2.5/5 — Low.** New ML training platform is well-architected but tests are integration-only, LLM-dependent, and cover happy-path only. Core multi-agent workflows still a black box.

---

## DATA-Athena-Snowflake — Architecture

**Pattern:** Stimulus-Response multi-agent orchestration via LangGraph

Request → StimulusRegistry → OrchestratorWorkflow → Branch (DAG of Nodes)  
 → AgentWorkflow (LangGraph StateGraph) → Response

**11 registered stimuli (workflows)**, each with versioned implementations (e.g. `user_question_submitted` at v4.3/4.4/4.5). Each branch is a `BaseBranch` subclass composing `AgentStrategy` (LLM nodes) and `ToolStrategy` (deterministic tools) into a LangGraph DAG.

**Key components:** - `OrchestratorWorkflow` — dynamically builds workflow from YAML config + registry - `StimulusRegistry` — central map of stimulus → class, version, alias - `AgentWorkflow` — LangGraph `StateGraph` wrapper with `MemorySaver` checkpointing - `MessagesState` — carries full conversation history between nodes - FastAPI middleware stack: Auth → ValidateUsage → LogRequest → ApplyUsage → SessionLifecycle - LLM backends: OpenAI GPT-4o (default), AWS Bedrock (Claude 3.7 Sonnet, Amazon Nova Micro), Snowflake LLM

**Notable architectural gaps:** - Strategy Director matcher + ranker nodes have placeholder code — not functional (`exit()` in matcher, dummy prompts in ranker) - No retry-specific workflow — `retry_optimization_requested` stimulus is entirely missing (P-05 gap) - No traditional ML — all inference is LLM-based with hardcoded thresholds (e.g. 15% acceptance drop) - No circuit breaker — a cascading failure brings down the entire workflow - OpenTelemetry integration is commented out — no observability in production

---

## athena-platform — Testing Coverage

Metric	Detail
Test files ( <code>_test.go</code> )	126
Test functions	777
Test frameworks	Go <code>testing</code> + testify (assert/require/mock), in-memory SQLite for repos
CI	GitHub Actions on every PR — <code>make test/coverage</code>
Coverage threshold	<b>20%</b> (comment in code: “TODO: raise to 65”)
<code>internal/clients/</code>	<b>Excluded from coverage entirely</b>

**Well tested:** - All 44 domain service interfaces have test files - ~43 PostgreSQL repository implementations tested via in-memory SQLite - V1 REST handlers: 15/18 (83%) - Auth middleware (JWT + API key) tested - Snowflake, Hermes, Merchant clients tested

**Critical gaps (untested):** - **V2 API: 0/18 handlers** — entire new API version has zero test coverage - **Bedrock client: 0%** — ML inference client excluded from coverage, no tests - 4 domain services untested: **auth, bedrock, element, workspace** - Bootstrap/DI integration test skipped (TODO: testcontainers) - 3 V1 handlers untested: **agent, workspaces, elements** - 0 benchmark tests — no performance regression safety net

**Maturity: 3.5/5** — **Solid foundation, with a critical blind spot in v2 + ML inference path.**

---

## athena-platform — Architecture

**Pattern:** Clean Architecture — strict layering

REST Handlers (v1 / v2, Gin)

↓

Controllers (~30 implementations)

↓

Domain Services (44 domain packages)

↓

Repositories (43 GORM implementations)

↓

PostgreSQL (RDS Multi-AZ) + Redis (ElastiCache)

**Key design choices:** - Constructor injection throughout — all services receive interface dependencies - 43 repository implementations, each domain entity has its own repo with query builders - Interface-driven — testify mocks and custom fake Redis for unit isolation - Event outbox table (**athia\_event\_outbox**) for reliable event delivery - Custom error types with HTTP status code mapping

**Experiment / A-B testing system:** - 4 model types: **processor\_selector, retry\_predictor, retry\_sequence, installment\_optimizer** - Experiment assignment: `SHA256(transaction_id) % 10000` — deterministic bucketing - Redis-cached assignments with 24h TTL - Auto-winner worker (`cmd/worker/`) evaluates statistical significance → force-routes all traffic to winner - Rich stratification context: device, geo, card BIN, merchant tier, timing (hour/day/payday windows), customer LTV

**Feedback loop:** - Merchants POST feedback on predictions → PostgreSQL → feeds retraining signals via `SendFeedback()` / `SendBatchFeedback()`

**Notable architectural gaps:** - V2 API exists but fully untested — suggests incomplete refactoring - Bedrock client (ML path) untested and excluded from coverage config - No event-driven cache invalidation — stale experiment assignments possible for up to 24h after config changes - Tight coupling to `*gin.Context` — hard to test handlers without HTTP server - Experiment context passed as ad-hoc parameters — no middleware to propagate it automatically

---

## Testing Summary — Both Repos

	DATA-Athena-Snowflake	athena-platform
Language	Python / FastAPI / LangGraph	Go / Gin
Test count	450+ functions, 36 files	777 functions, 126 files
Estimated coverage	~20%	~25–30%
Core product tested?	No — multi-agent workflows untested; new ML tests need live deps	Partially — v2 API + Bedrock missing; Triton + v2 encoder tests merged
Architecture pattern	Stimulus-response / LangGraph DAG	Clean Architecture / Repository
Biggest risk	Multi-agent core is a black box	V2 API + ML inference path untested

	DATA-Athena-Snowflake	athena-platform
CI enforced?	Partial — fragmented workflows; black + isort improving	Yes — every PR
Version	main (feat/ATH-0000 merged) + feat/ATH-1024 (PR pending)	v0.15.5 (main); Triton merged
Maturity	<b>3/5</b> — <b>Improving</b> (ML service tests on feature branch)	<b>4/5</b> — <b>Solid</b> (Triton merged, 32+ new tests)

**Recommended immediate actions (both repos):** 1. Merge `feature/llm-driven-ml-training` (Triton IS) — unblocks G-06 completion and shadow mode deployment (athena-platform) 2. Wire `model_deployer.py` to call `athia-model-server` HTTP API — completes G-06 (~1.5d) (DATA-Athena-Snowflake) 3. Add tests for all 18 v2 handlers (athena-platform) 4. Add tests for Bedrock client + service (athena-platform) 5. Remove `internal/clients/` from coverage exclusions (athena-platform) 6. Raise coverage threshold from 20% → 60% with CI enforcement (athena-platform) 7. Build `retry_optimization_requested` stimulus — currently missing entirely (DATA-Athena-Snowflake)

---

### Open Questions Raised by Repo Analysis

#	Question	Owner	Status
Q-R1	Are ATHIA_PREDICTIONS and ATHIA_FEEDBACK tables already populated in Deuna's Snowflake, or only in Athia's internal Snowflake?	Israel / Rakesh	<b>Confirmed</b> (2026-02-24)
Q-R2	Are SageMaker endpoints currently live for <code>processor_selector / retry_predictor</code> , or are they placeholders?	Rakesh	Open
Q-R3	Is there a working model in <code>MODEL_ARTIFACTS</code> that Deuna's payment service is actually calling today?	Rakesh	Open
Q-R4	What is the current payment volume through the routing engine? (Needed to validate sample size requirements for A/B tests)	Israel	Open
Q-R5	Who owns the athena-platform Go repo deployment? Aidaptive or Deuna infra?	Pablo / Rakesh	Open
Q-R6	When will <code>feature/llm-driven-ml-training</code> (Triton IS) merge to main? This determines G-06 completion timeline and whether Triton or SageMaker is the production model serving backend.	Pablo / Rakesh	Open

---

## Volaris Smartrouting — Delivery Task List

**Scope:** End-to-end implementation of AI-powered smartrouting for Volaris (Phase 1 client). **Processors:** Worldpay (ID:76), MIT (ID:85), Elavon (cards), Amex (Amex cards only) **Use Cases:** P-01 (outage detection), P-02 (routing optimizer), P-03 (per-transaction route selection), P-04 (message manipulation), P-05 (retry optimization) **Success target:** Measurable approval rate lift, PSP outage stability, p95 latency < 200ms

---

### Phase 0 — Service Architecture & Shell Setup

**Goal:** Define service boundaries, API contracts, and scaffold all 7 service shells using the TensorFlow ecosystem. This phase establishes the ML infrastructure foundation before any Volaris-specific work begins.

---

#	Task	Notes	Effort	Status
V-D01	Design overall service architecture — define 7 service boundaries, data flow, inter-service communication patterns	Rakesh — architecture doc with service dependency graph	2d	[~]
V-D02	Define API contracts for all 7 services — request/response schemas, error handling, versioning strategy	Rakesh — OpenAPI specs per service	1.5d	[~]
V-D03	Design TensorFlow ecosystem integration plan — map TFX components to services, define TF Serving model format, TFDV/TFMA integration points	Rakesh — tech spec	1d	[~]
V-S01	Scaffold Data Pipeline service shell — TFX ExampleGen + StatisticsGen, Snowflake ingestion adapter, TFDV schema generation	TensorFlow Data Validation for automated data checks	1.5d	[~]
V-S02	Scaffold Feature Service shell — TF Transform <code>preprocessing_fn</code> , feature store API, real-time serving endpoint, feature versioning	FeatureService built on feat/ATH-1024 branch (CRUD, groups, lineage, extraction, tests, UI)	1.5d	[~]
V-S03	Scaffold Training Pipeline service shell — TFX Trainer with <code>tf.keras</code> models, hyperparameter tuning via Keras Tuner, training history tracking	TFX pipeline with 9 components + penguin demo built on feature branch	1.5d	[~]

#	Task	Notes	Effort	Status
V-S04	Scaffold Model Management service shell — model registry CRUD, artifact storage (SavedModel format), lifecycle states, version comparison	ModelArtifactService built on feat/ATH-1024 branch (CRUD, versioning, tagging, tests, UI)	1d	[~]
V-S05	Scaffold Eval Service shell — TFMA integration, per-slice metrics, regression detection, model blessing/rejection API	EvaluationService built on feature branch (inference pipeline, analytics, tests, UI)	1d	[~]
V-S06	Scaffold Evaluation Framework shell — A/B test statistical engine, experiment winner detection, guardrails (latency, revenue), significance calculator	Extends existing athena-platform auto-winner logic	1.5d	[]
V-S07	Scaffold Experiment System shell — experiment CRUD, traffic splitting, variant management, shadow mode orchestration, merchant-scoped experiments	ClearML experiment tracking built on feature branch (offline log reader, comparison UI, e2e tests)	1.5d	[~]
V-S08	Set up shared TensorFlow dependencies — tensorflow, tfx, tensorflow-transform, tensorflow-model-analysis, tensorflow-data-validation, keras-tuner	TFX deps + Docker base image built on feature branch; uv migration by Naoki	0.5d	[~]

**Phase 0 total: ~14.5d**

**Owner:** V-D01 through V-D03 are **Rakesh** (architecture & design). V-S01 through V-S08 are engineering tasks for the team.

### Phase 1 — Discovery & EDA

**Goal:** Deeply understand Volaris transaction data before building anything. All decisions in later phases depend on this.

#	Task	Snowflake Source	Effort	Status
V-01	Filter all Volaris transactions from <code>ABTESTING.ALL_VIEWS_FLAT</code> — establish date range, total volume, and monthly trend	<code>ABTESTING.ALL_VIEWS_FLAT</code>	0.5d	[]
V-02	Compute per-processor approval rates for Volaris: Worldpay (76), MIT (85), Elavon, Amex — identify which processor performs best by card type, currency, amount band	<code>VW_ATHENA_PAYMENT_ATTEMPT</code>		[]
V-03	Analyze retry patterns — how many attempts per order, which processor was retried to, success rate on 1st vs 2nd vs 3rd attempt	<code>VW_ATHENA_PAYMENT_ATTEMPT</code> ( <code>PAYMENT_ATTEMPT_SEQUENCE_ORDER</code> , <code>PAYMENT_LAST_ATTEMPT_INDICATOR</code> )		[]
V-04	Explore <code>DYNAMIC_ROUTING_DETAIL</code> JSON column — extract all keys/values to understand what routing metadata Deuna already captures per attempt	<code>VW_ATHENA_PAYMENT_ATTEMPT</code>		[]
V-05	Map Volaris routing rules — query <code>VW_ROUTING_MERCHANT_RULE</code> , <code>VW_ROUTING_MERCHANT_RULE_CONDITION</code> , <code>VW_ROUTING_MERCHANT_RULE_MEMBER</code> for Volaris merchant ID	<code>SOURCES</code> schema	0.5d	[]
V-06	Analyze existing smart routing log for Volaris — which algorithm types are used, skip rates, process time distribution (validate p95 latency baseline)	<code>VW_SMART_ROUTING_ATTEMPTS</code>		[]
V-07	Analyze hard vs. soft decline distribution by processor and error code — distinguish recoverable vs. terminal failures	<code>VW_ATHENA_PAYMENT_ATTEMPT</code> ( <code>HARD_SOFT</code> , <code>ERROR_CODE</code> , <code>ERROR_CATEGORY</code> )		[]
V-08	Profile Volaris-specific airline features — flight routes, passenger count, booking window — assess predictive signal for routing	<code>VW_ORDER_AIRLINE_DETAIL_ALL</code> , <code>VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL</code>		[]

#	Task	Snowflake Source	Effort	Status
V-09	Assess sample size adequacy for A/B testing — compute daily transaction volume per processor; validate 1000 samples/variant is achievable within 7 days	ABTESTING.ALL_VIEWS_FLAT	1d	[]
V-10	Produce EDA summary report — approval rates, retry rates, error code taxonomy, processor share, feature correlations	All above	1d	[]

**Phase 1 total: ~7.5d**

## Phase 2 — Feature Engineering

**Goal:** Build a clean, versioned feature set for Volaris ML models. Features must be computable at inference time (< 200ms) and reproducible for retraining.

#	Task	Notes	Effort	Status
V-11	Define Volaris feature schema — list all features, data types, source tables, and compute latency	Covers card BIN/brand/bank, amount, currency, CIT/MIT type, retry count, error history, user RFM, geo, device, time-of-day	1d	[]
V-12	Build card-level features — BIN, brand (Visa/MC/Amex), issuing bank, card type, country; pre-compute historical approval rate per BIN per processor	ABTESTING.ALL_VIEWS_FLAT		[]
V-13	Build transaction-level features — amount bucket, currency, CIT vs MIT, MCC, order type (flight, hotel, ancillary)	VW_ATHENA_PAYMENT_ATTEMPT, VW_ORDER_AIRLINE_DETAIL_ALL		[]
V-14	Build user-level features — RFM scores, fraud rate cohort, tenure, browsing session signals, past approval/decline rate	ABTESTING.ALL_VIEWS_FLAT		[]

#	Task	Notes	Effort	Status
V-15	Build retry-context features — previous processor used, previous error code, time since last attempt, attempt number	VW_ATHENA_PAYMENT_ATTEMPT		[]
V-16	Build processor-state features — rolling approval rate per processor (15-min, 1-hour, 24-hour windows); soft decline rate; timeout rate for P-01	VW_SMART_ROUTING_ATTEMPT		[]
V-17	Amex special-case logic — Amex card transactions must always route to Amex processor; build hard rule + bypass ML for this path	Logic layer	0.5d	[]
V-18	Build training dataset — join all features onto labeled outcomes (approved=1, declined=0) per payment attempt; split into train/validation/test sets	ATHIA_TRAINING_DATASET1d or custom SQL		[]
V-19	Validate feature quality — check null rates, distribution skew, leakage risk, correlation with outcome	Feature audit	1d	[]

**Phase 2 total: ~8.5d**

### Phase 3 — Model Development (P-03 & P-05)

**Goal:** Train and evaluate `processor_selector` and `retry_predictor` models for Volaris's 4 processors using the TensorFlow ecosystem.

#	Task	Notes	Effort	Status
V-20	Train <code>processor_selector</code> v1 — rank Worldpay, MIT, Elavon, Amex by predicted approval probability	<code>tf.keras</code> DNN; input: Phase 2 TF Transform features; label: approved by processor; export as SavedModel	2d	[]

#	Task	Notes	Effort	Status
V-21	Evaluate <code>processor_selector</code> — AUC, approval lift vs. static rules, per-processor accuracy, TFMA per-slice analysis	TFMA evaluation + Eval Service	1d	[]
V-22	Train <code>retry_predictor</code> v1 — predict approval probability on retry given error code, processor, retry count, time elapsed	<code>tf.keras</code> binary classifier; TFX Trainer component	1.5d	[]
V-23	Train <code>retry_sequence</code> v1 — predict optimal processor order for retry (e.g. if Worldpay failed, try MIT vs. Elavon first)	<code>tf.keras</code> ranking/multi-class model	1.5d	[]
V-24	Evaluate retry models — retry success rate lift vs. current behavior; TFMA slice analysis for processor fatigue	Eval Service + TFMA per-slice metrics	1d	[]
V-25	Architecture comparison for <code>processor_selector</code> — DNN vs. wide-and-deep vs. gradient-boosted trees (TF Decision Forests); select champion	TF ecosystem algorithm comparison; replaces XGBoost vs. LR comparison	1d	[]
V-26	Implement inference latency test — ensure all TF Serving models serve predictions under 50ms (headroom for p95 < 200ms)	TF Serving benchmark via Eval Service	0.5d	[]
V-27	Package models — export SavedModel artifacts, document input schema, feature versions, TFMA metrics	Model Management service handles artifact storage	0.5d	[]

**Phase 3 total: ~9d**

#### Phase 4 — Outage Detection & Failover (P-01)

**Goal:** Detect PSP failures in real time and auto-fail-over to an alternate Volaris processor.

#	Task	Notes	Effort	Status
V-28	Define outage signal — persistent timeout/error codes that indicate a processor is down (not just a single declined transaction)	Based on V-07 error code taxonomy; set window + threshold	1d	[]
V-29	Implement rolling processor health score — compute per-processor timeout rate and soft decline rate over a sliding 5–15 min window	Uses <code>VW_SMART_ROUTING_ATTEMPTS</code> real-time feed	1.5d	[]
V-30	Implement failover logic — when health score drops below threshold, automatically skip affected processor and route to next-best available Volaris PSP	In athena-platform routing layer	1.5d	[]
V-31	Implement recovery detection — random-sample downed PSP at low rate (e.g., 1–2% of traffic) to detect when it recovers; auto-restore on consecutive successes	Prevents permanent blacklisting	1d	[]
V-32	Test outage simulation — inject synthetic failures for each of the 4 Volaris processors; verify failover and recovery behave correctly	Integration test	1d	[]
V-33	Add alerting — Slack/PagerDuty notification when a Volaris processor is flagged as down or recovered	athena-platform observability	0.5d	[]

**Phase 4 total: ~6.5d**

---

### Phase 5 — Message Manipulation (P-04)

**Goal:** Test toggling CIT/MIT, AVS, and MCC fields in authorization messages to improve approval rates.

#	Task	Notes	Effort	Status
V-34	Audit current CIT/MIT usage for Volaris — query MCI_MSI_TYPE, ORDER_MCI_MSI_TYPE, PAYMENT_ATTEMPT_METHOD_TYPE to understand today's distribution	ABTESTING.ALL_VIEWS_FLAT	1d	[]
V-35	Identify approval rate delta by CIT vs MIT per processor — determine if toggling improves approval on specific Volaris PSPs	EDA + statistical test	1d	[]
V-36	Design message manipulation experiment — define which combinations to test (CIT/MIT × processor × card type)	Requires Deuna eng coordination	1d	[]
V-37	Implement message manipulation API — endpoint in athena-platform to recommend CIT/MIT, AVS, MCC setting per transaction	New athena-platform endpoint	2d	[]
V-38	Run A/B test — compare approval rates with vs. without recommended message changes for Volaris	Via existing A/B testing infrastructure	1d	[]

**Phase 5 total: ~5.5d**

## Phase 6 — athena-platform Integration

**Goal:** Register Volaris models, create experiment, connect prediction API to Deuna's payment service.

#	Task	Notes	Effort	Status
V-39	Register <code>processor_selector</code> model artifact for Volaris in <code>MODEL_ARTIFACTS</code> — include version, Triton backend ref, feature schema	POST <code>/api/v1/ml/models</code> (new ExperimentService API)	0.3d	[]
V-40	Register <code>retry_predictor</code> and <code>retry_sequence</code> model artifacts for Volaris	Same — ExperimentService handles all model types	0.3d	[]

#	Task	Notes	Effort	Status
V-41	Create Volaris-scoped experiment — merchant ID filter, traffic split (10% treatment), shadow mode validation, guardrails	POST <code>/api/v1/ml/experiments</code> with variants in one call (Triton branch)	0.5d	[]
V-42	Validate experiment assignment — confirm SHA256 bucketing is deterministic for Volaris transaction IDs; test control vs. treatment assignment	End-to-end test	0.5d	[]
V-43	Coordinate with Deuna engineering — define API contract for Deuna payment service to call POST <code>/api/v1/ml/predict</code> with Volaris transaction context	Deuna eng (Pablo/Israel)	1d	[]
V-44	Implement Deuna payment service integration — Deuna calls athena-platform at routing decision point; receives ranked processor list; applies to routing	Deuna-side work + athena validation	2d	[]
V-45	End-to-end integration test — send test Volaris transactions through full flow: Deuna → athena-platform → model → ranked processors → routing	Staging environment	1d	[]
V-46	Shadow mode validation — run model in shadow mode (log predictions, don't act on them) for 48h; compare predicted vs. actual routing outcomes	Built-in <code>is_shadow_mode=true</code> in Triton branch; set up + monitor only	0.5d	[]

**Phase 6 total: ~6.5d** (updated from 7.5d — savings from *ExperimentService* one-call API and built-in shadow mode)

## Phase 7 — Monitoring & Feedback Loop

**Goal:** Ensure models degrade gracefully, outcomes flow back into retraining, and the team has visibility.

#	Task	Notes	Effort	Status
V-47	Confirm ATHIA_PREDICTIONS and ATHIA_FEEDBACK tables are active in Deuna's Snowflake — verify data is flowing from live predictions	Israel / Rakesh	0.5d	[x]
V-48	Deploy ATHIA_STAGE_OUTCOMES table — needed to track per-stage funnel (routing decision → outcome → retry → final result)	Table created in feat/ATH-0000 branch; needs deployment to Deuna production Snowflake	0.5d	[~]
V-49	Set up Volaris approval rate dashboard — daily and hourly approval rate per processor, vs. pre-Athia baseline	Grafana or Snowflake SQL	1d	[]
V-50	Set up model performance dashboard — prediction confidence distribution, processor rank accuracy, retry success rate lift	Grafana CloudWatch dashboard merged (ML latency, confidence, backend distribution); needs Volaris-specific panels	1d	[~]
V-51	Define retraining trigger — set thresholds: if approval rate drops > X% or model AUC drops > Y%, trigger retraining	Policy document + automated trigger	0.5d	[]
V-52	Schedule retraining pipeline — weekly or bi-weekly retraining run using latest ATHIA_TRAINING_DATASET data; auto-register new version	Orchestration (maps to G-01, G-02)	1d	[]
V-53	Implement auto-winner evaluation for Volaris experiment — confirm auto-winner worker runs for Volaris experiment with correct guardrails	athena-platform worker	0.5d	[]
V-54	Post-launch review — after 2 weeks live, analyze lift in approval rate, outage response time, retry success rate; document findings	Rakesh + Pablo	1d	[]

**Phase 7 total: ~6d**

## Volaris Task Summary

Phase	Focus	Tasks	Effort
<b>0 — Service Architecture</b>	<b>Design + scaffold TensorFlow service shells</b>	<b>11</b>	<b>14.5d</b>
1 — Discovery & EDA	Understand Volaris data	10	7.5d
2 — Feature Engineering	Build ML features (via Feature Service + TF Transform)	9	8.5d
3 — Model Development	Train processor_selector + retry models (tf.keras)	8	9d
4 — Outage Detection	P-01 failover for 4 PSPs	6	6.5d
5 — Message Manipulation	P-04 CIT/MIT experiment	5	5.5d
6 — Platform Integration	Register models, wire Deuna	8	<b>6.5d</b>
7 — Monitoring & Feedback	Dashboards, retraining, review	8	6d
<b>Total</b>		<b>65 tasks</b>	<b>~64d</b>

### Timeline Estimate

Scenario	Duration
3 engineers (parallel where possible)	<b>5–6 weeks</b>
2 engineers	<b>7–8 weeks</b>
1 engineer	<b>~13 weeks</b>

**Note:** Phase 0 runs first (Rakesh designs in parallel with team scaffolding). Phases 1–2 sequential (EDA before features). Phases 3–5 can run in parallel after Phase 2. Phase 6 requires Deuna engineering coordination — plan for 1-week lead time. **ML Stack:** TensorFlow ecosystem — `tf.keras`, TFX, TF Serving, TFDV, TFMA, TF Transform, Keras Tuner. Replaces Snowflake ML / XGBoost / scikit-learn.

### Codebase Improvement Suggestions

**Goal:** Bring both repos from their current state to production-grade quality. **Analyzed:** 2026-02-19. Prioritized by impact and urgency.

#### DATA-Athena-Snowflake — Suggestions

##### Testing (Critical)

#	Suggestion	Priority	Effort
T1	Add pytest unit tests for all 14 route handlers — use <code>TestClient</code> from FastAPI + mock services	Critical	3d
T2	Add unit tests for all 13 service layer files — mock Snowflake sessions and external clients	Critical	3d

#	Suggestion	Priority	Effort
T3	Add unit tests for core multi-agent framework: <b>AgentWorkflow</b> , <b>AgentStrategy</b> , <b>ToolStrategy</b> node/edge composition	Critical	2d
T4	Add per-branch workflow tests for all 11 stimuli (at minimum the default version of each) — mock LLM responses with deterministic fixtures	High	4d
T5	Add unit tests for Redis, Postgres, and Platform clients	High	1d
T6	Add unit tests for all 4 Lambda / AgentCore entrypoints	High	1d
T7	Unify CI into a single <b>pytest</b> run covering all domains — replace fragmented per-domain workflows	High	1d
T8	Enable <b>pytest-cov</b> with a minimum 60% threshold enforced in CI	High	0.5d
T9	Migrate <code>/test/api/</code> manual scripts to proper <b>pytest</b> integration tests	Medium	2d
T10	Add <code>@pytest.mark.asyncio</code> coverage for all async FastAPI route handlers	Medium	1d

## Architecture (High)

#	Suggestion	Priority	Effort
A1	Build <code>retry_optimization_requested</code> stimulus — new LangGraph branch with dedicated retry analysis nodes; maps to P-05	Critical	3d
A2	Complete Strategy Director: replace <code>exit()</code> placeholder in <code>Matcher</code> and dummy prompts in <code>Ranker</code> with real logic	Critical	2d
A3	Add circuit breaker pattern to <b>AgentWorkflow</b> — isolate node failures so one failing agent doesn't crash the entire DAG	High	2d
A4	Enable OpenTelemetry tracing — currently commented out in <code>main.py</code> ; add trace IDs across all workflow nodes	High	1.5d

#	Suggestion	Priority	Effort
A5	Replace hardcoded thresholds (e.g. 15% acceptance drop, 60–80 min fraud windows) with configurable, data-driven parameters	High	2d
A6	Standardize tool definition — consolidate <code>@create_tool</code> decorator vs. manual tool definitions; add tool versioning	Medium	1d
A7	Add configuration management layer — replace scattered <code>load_dotenv</code> with a single validated config schema (Pydantic Settings)	Medium	1d
A8	Add global rate limiting middleware — currently only per-user/per-feature quotas exist	Medium	0.5d
A9	Implement LLM prompt injection guards in all system prompts — audit and sanitize user-controlled inputs before they reach LLM	High	1d
A10	Document each stimulus branch architecture with up-to-date README and flow diagram (many are stale or missing)	Medium	1d

## athena-platform — Suggestions

### Testing (Critical)

#	Suggestion	Priority	Effort
T1	Add tests for all 18 V2 REST handlers — entire new API version has 0% coverage	Critical	4d
T2	Add tests for Bedrock client ( <code>/internal/clients/bedrock/</code> ) and Bedrock domain service	Critical	1.5d
T3	Remove <code>internal/clients/</code> from coverage exclusions in <code>/scripts/check-coverage.sh</code>	Critical	0.5d
T4	Raise coverage threshold from 20% → 60% in CI; add enforcement (fail build below threshold)	High	0.5d
T5	Implement bootstrap integration test using <code>testcontainers-go</code> — currently skipped due to Redis dependency	High	1.5d

#	Suggestion	Priority	Effort
T6	Add tests for the 3 untested V1 handlers: <code>agent</code> , <code>workspaces</code> , <code>elements</code>	High	1d
T7	Add tests for 4 untested domain services: <code>auth</code> , <code>bedrock</code> , <code>element</code> , <code>workspace</code>	High	1.5d
T8	Add benchmark tests ( <code>func Benchmark...</code> ) for high-traffic endpoints: <code>/ml/predict</code> , <code>/feedback</code> , experiment assignment	Medium	1d
T9	Replace in-memory SQLite with <code>testcontainers-go</code> PostgreSQL for JSON aggregation tests (currently skipped)	Medium	1.5d
T10	Add contract tests for Snowflake and Bedrock APIs — validate request/response shapes don't drift silently	Medium	2d

## Architecture (High)

#	Suggestion	Priority	Effort
A1	Add event-driven cache invalidation for model registry — when an experiment config changes, publish an event that invalidates Redis cache immediately (currently 24h TTL with no invalidation)	High	1.5d
A2	Add experiment context propagation middleware — automatically enrich request context with session/experiment metadata rather than manual enrichment per handler	High	2d
A3	Abstract HTTP request/response away from <code>*gin.Context</code> in controllers — use a transport-agnostic interface; makes handlers testable without an HTTP server	High	2d
A4	Implement domain events for feedback and telemetry — replace HTTP POST feedback with event-driven publishing (leverage existing <code>athia_event_outbox</code> table)	Medium	2d

#	Suggestion	Priority	Effort
A5	Deploy <del>ATHIA_STAGE_OUTCOMES</del> and <del>ATHIA_SESSION_SUMMARY</del> Snowflake tables — Done in feat/ATH-0000 branch	High	1d 0d
A6	Add global rate limiting middleware — no global limiter exists, only per-user/feature quotas	Medium	0.5d
A7	Implement structured logging — standardize log fields across all handlers using the existing logger pkg; add request ID, merchant ID, experiment ID to all logs	Medium	1d
A8	Merge <del>feature/llm-driven-ml-training</del> (Triton IS) to main — completes G-06, adds shadow mode, ExperimentService one-call API, model readiness checks	High	0.5d
A9	Set up production Grafana dashboards + alert rules — Grafana config exists locally (docker-compose) but no production dashboards or alerts are defined	High	2d
A10	Add versioned API deprecation headers to V1 responses — signal to consumers that V2 is the preferred version	Low	0.5d

### Combined Priority Order

Priority	Action	Repo	Effort
Critical	Build retry optimization stimulus (P-05 gap)	DATA-Athena-Snowflake	3d
Critical	Complete Strategy Director (P-02 gap)	DATA-Athena-Snowflake	2d
Critical	Add tests for V2 REST API (18 handlers)	athena-platform	4d
Critical	Add tests for Bedrock client + service	athena-platform	1.5d
Critical	Add route + service + client tests	DATA-Athena-Snowflake	7d
Critical	Remove <code>internal/clients/</code> from coverage exclusions	athena-platform	0.5d
High	Add multi-agent framework + branch tests	DATA-Athena-Snowflake	6d
High	Add circuit breakers to AgentWorkflow	DATA-Athena-Snowflake	2d
High	Enable OpenTelemetry tracing	DATA-Athena-Snowflake	1.5d
High	Raise CI coverage threshold to 60%	athena-platform	0.5d
High	Bootstrap integration test (testcontainers)	athena-platform	1.5d
High	Event-driven cache invalidation	athena-platform	1.5d
High	Deploy ATHIA_STAGE_OUTCOMES + SESSION_SUMMARY	athena-platform	1d

Priority	Action	Repo	Effort
High	Model warm-up for SageMaker cold starts	athena-platform	1d
Medium	Adaptive thresholds (replace hardcoded values)	DATA-Athena-Snowflake	2d
Medium	Experiment context middleware	athena-platform	2d
Medium	Production Grafana dashboards + alerts	athena-platform	2d
Medium	Benchmark tests for hot endpoints	athena-platform	1d
Medium	Unified CI suite + coverage enforcement	DATA-Athena-Snowflake	1.5d

## Notes & Meeting Log

### 2026-03-16

- **Repo sync:** Both repos on `main`. DATA-Athena-Snowflake: 27 new commits since last sync (172 files, +25.6K/-3.5K). athena-platform: 27 new commits (v0.15.0 → v0.15.5).
- **DATA-Athena-Snowflake (main):** ATH-1136 ML/LLM ingestion pipeline merged. ATHIA\_PREDICTIONS/FEEDBACK schema extensions. Snowflake session pool isolation. Acceptance Rate Analyzer v2. ML doc reorganization.
- **DATA-Athena-Snowflake (feature branch feat/ATH-1024-model-artifact-management):** 184 files, +36K. Model Artifact Service, Feature Service, Evaluation Service, TFX Pipeline, ClearML Experiment Tracking. PR pending review.
- **athena-platform (main v0.15.0→v0.15.5):** Shadow mode merged. OTEL metrics pipeline. Grafana CloudWatch dashboard. V2 processor selector encoder with dynamic per-model encoders. A/B model version routing. Prediction event handling with flexible metadata.
- **Team activity:** Rakesh ~100+ commits (all 5 ML platform services). Naoki: uv migration, test fixtures. Kedar: Snowflake feature extraction exploration. Rene: first model work.
- **Gap changes:**
  - G-03 (CI/CD): Not started → **Partial** (CI improvements: black, isort, uv, pytest fixtures)
  - G-05 (Monitoring): Partial → **Nearly Done** (OTEL + Grafana on main)
  - G-08 (Feature Store): Partial → **Partial+** (v2 encoder feature mappings on main)
  - G-13 (Versioning): Partial → **Nearly Done** (dynamic encoders + A/B version routing on main)
- **Effort delta:** Total saved ~75.5d (was ~73d, +2.5d). Remaining ~29d (was ~31.5d, -2.5d).
- **Volaris Phase 0:** 10 of 11 tasks now [~] partial — service shells being built on feature branches.

### 2026-03-13

- **Architecture decision:** Adopted **TensorFlow ecosystem** (TFX, TF Serving, TFDV, TFMA, TF Transform) for all ML work. Replaces Snowflake ML / XGBoost / scikit-learn approach.
- **Service architecture:** Defined 7 service shells to implement: Data Pipelines, Feature Service, Training Pipelines, Model Management, Eval Service, Evaluation Framework, Experiment System.
- **Task list expanded:** Added Phase 0 (Service Architecture & Shell Setup) with 11 new tasks (V-D01–V-D03 design, V-S01–V-S08 scaffolding). Total now 65 tasks, ~64d.
- **Design ownership:** Rakesh owns V-D01 (service architecture), V-D02 (API contracts), V-D03 (TF ecosystem integration plan).
- **Repo sync:** Both repos now on `main` branch. DATA-Athena-Snowflake `feat/ATH-0000` branch merged to `main` (182 files, +29K lines). athena-platform already on `main` (v0.15.5).
- **Critical finding:** Training pipeline is now on `main` in DATA-Athena-Snowflake. All ML services now available on `main`.
- **Both repos fully merged:** No more feature branch tracking needed. All work is on `main`.
- **New on main (DATA-Athena-Snowflake):** 13 new services in `src/services/`, 8 new route files, 14 ML docs in `docs/ml/`, Snowflake ML tables, ingestion API, feedback routes
- **API endpoints now on main:** training plan/run/decision, experiment design, model registry CRUD, feedback collection, schema discovery

## 2026-03-12

- **Repo sync:** athena-platform pulled 27 new commits (395f1c0 → 7aeedb9, v0.15.0 → v0.15.5). DATA-Athena-Snowflake unchanged.
- **Critical finding:** feature/llm-driven-ml-training branch fully merged to main. Triton Inference Server integration, ExperimentService API, shadow mode, Grafana dashboard, processor selector v2 — all now on main.
- **New feature branches detected:** feat/new-encoder-v2 (minor), feature/experiment-api-cleanup (58 files, in review)
- **Gap changes:**
  - G-05 (Monitoring): Metrics instrumented + Grafana CloudWatch dashboard. Saved ~4d (was ~2d)
  - G-06 (Deployment): Triton integration merged. Saved ~6d, only ~1.5d remaining
  - G-13 (Versioning): Auto version + variant deprecation merged. Saved ~4d (was ~2.5d)
  - G-14 (Rollback): Shadow mode + is\_default merged. Saved ~4d (was ~2.5d)
- **Effort delta:** Total saved ~73d (was ~60d, +13d). Remaining ~31.5d (was ~45d, -13.5d)
- **Testing:** 22 new test files added (Go + Python). athena-platform coverage increased significantly.
- **Processor Selector v2:** 54-feature XGBoost encoder production-ready. v1 and v2 models both in repo.

## 2026-02-26

- **Synced both repos to latest and ran full re-analysis.**
- **DATA-Athena-Snowflake (feat/ATH-0000):** One new commit — minor fix to llm\_training\_orchestrator.py model configs + DISABLE\_AUTH env var support in auth.py. No new capabilities.
- **athena-platform now at v0.14.0** (two releases since last analysis: v0.13.0 and v0.14.0).
  - **AgentCore VPC endpoint support** — Bedrock can now route through private VPC for secure connectivity (critical for production).
  - Strategy display\_order field added; strategy metadata enhancements (ATH-1110); X-Timezone CORS header.
  - New DB migrations: extended varchar lengths, display\_order column.
- **Critical new finding — feature/llm-driven-ml-training branch (athena-platform, in review):**
  - **Triton Inference Server** full integration: athia-model-server sidecar extended with TritonClient (gRPC), ModelConversionManager (sklearn→ONNX→Triton), ModelManager.
  - **Shadow mode experiments:** is\_shadow\_mode + is\_default flags on model\_experiments. All 3 model types (processor\_selector, installment\_optimizer, retry\_predictor) seeded with default shadow mode experiments.
  - **ExperimentService** — new service: create experiment + variants + models in one atomic POST /api/v1/ml/experiments call; max\_variants constraint (default: 3); variant deprecation by ID.
  - **ServiceWithTriton** — wraps model registry with Triton readiness checks; CheckExperimentReadiness API prevents routing to unconverted models.
  - **End-to-end deployment workflow documented** in TRITON\_MODEL\_DEPLOYMENT\_WORKFLOW.md — clear path from DATA-Athena-Snowflake training pipeline → EFS → athia-model-server → Triton → athena-platform experiment.
  - **32/32 new tests passing** in this branch.
- **Gap status updates from Triton branch:**
  - G-06 (Deployment): Partial → **Nearly Done** — ~1.5d integration work remains (wire model\_deployer.py to call athia-model-server HTTP API).
  - G-14 (Rollback): Not started → **Partial** — shadow mode + is\_default = safe rollback mechanism.
  - G-13 (Versioning): More complete — max\_variants + variant deprecation + PROPER\_EXPERIMENT\_API\_DESIGN.md docs.
- **Updated effort estimates:**
  - Total saved: ~60d (was ~48d before Triton branch analysis).
  - Remaining: ~45d (was ~56.5d).
  - 2-engineer timeline: 5–6 weeks (was 7–8 weeks).
- **Volaris delivery impact:** Phase 6 effort reduced by ~1.5d (streamlined experiment API via ExperimentService, shadow mode built-in for V-46).

## 2026-02-20

- **Analyzed feat/ATH-0000-athia-ml-llm-schema-discovery branch of DATA-Athena-Snowflake.**
- This branch is a major capability expansion — the repo is no longer LLM-analytics only. It now includes a full ML training platform (~10,000 lines of new code).
- **15 new services** covering: model registry, schema discovery (ChromaDB), feature extraction, data quality validation (800+ lines), feedback collection (3 modes), LLM-powered training orchestration (GPT-4 + RAG), LLM experiment design (GPT-4 + RAG), training pipeline (Snowflake ML — LR, RF, XGBoost), model deployer, schema scheduler.
- **7 new route groups** covering training (plan/run/deploy/history), training decisions, model registry CRUD, schema discovery, event ingestion, feedback, and experiment design.
- **New SQL deployed in this branch:** ATHIA\_STAGE\_OUTCOMES and ATHIA\_SESSION\_SUMMARY (previously “designed, not deployed”) are now created. Also adds: ML\_MODEL\_REGISTRY, ATHIA\_TRAINING\_DATASET, ATHIA\_EXPERIMENT\_LIFT, ATHIA\_MULTI\_STAGE\_ANALYSIS, ATHIA\_MODEL\_METRICS.
- **Gap status updated:** G-01, G-04, G-07, G-09, G-11, G-12 now implemented. G-02, G-05, G-06, G-08, G-13 partially addressed. G-03, G-10, G-14 still not started.
- **Production readiness:** ~85%. Core training works. Remaining gap: athena-platform API integration (canary creation, model registration) is still manual.
- **Security issue noted:** SQL injection risks in multiple new services (string interpolation in schema\_discovery.py, training\_planner.py, athia\_ingestion.py, acceptance\_rate v1\_0 branch.py). Needs parameterized queries before production.
- Branch docs: 10 markdown files in docs/ cover all services in detail.

## 2026-02-19 (late)

- **Portal live at <https://deuna-ebce2.web.app>** — Google Auth gate deployed. Only @aidaptive.com and @deuna.com email addresses can sign in. Built on Firebase Hosting + Firebase Auth JS SDK (signInWithPopup).
- Favicon added (D×A SVG, hosted at /favicon.svg).
- Scoping project GitHub repo link removed from portal References section (internal repo — not for client view).
- Note: `signInWithRedirect` was tried as an alternative to fix a COOP console warning, but it broke the sign-in flow and was reverted back to `signInWithPopup`. COOP warning is cosmetic and does not affect functionality.
- v12 PDF export generated and deployed to hosting portal.

## 2026-02-19 (afternoon)

- **Merchant selection: Volaris chosen as the target merchant for Phase 1** (over Cinépolis).
  - Cinépolis shows only Cybersource (a gateway — actual processor behind it is unknown), making it harder to work with.
  - Volaris has a clear, known set of PSPs:
    - \* **Worldpay** (Processor ID: 76)
    - \* **MIT** (Processor ID: 85)
    - \* **Elavon** — used for cards
    - \* **Amex** — used specifically for Amex cards
  - Volaris uses **4 processors total for cards**.
  - Routing policies exist for different currencies: MIT, Elavon, Worldpay handle different currency flows; Amex is dedicated to Amex card transactions.
  - This clarity makes Volaris the right starting point for P-03 (per-transaction route selection) and P-05 (retry optimization).

## 2026-02-19

- Analyzed both Deuna GitHub repositories in full: DATA-Athena-Snowflake and athena-platform.
- Key finding: athena-platform is a production-ready Go REST API with model registry, A/B testing, and auto-winner selection already built. The missing piece is the training pipeline.
- Key finding: DATA-Athena-Snowflake is an LLM-based analytics platform (not ML training). It generates strategies via GPT-4o / Claude, not trained models.

- The inference types `processor_selector` and `retry_predictor` already exist in the model registry schema — these map directly to P-03 and P-05.
- No retry-specific workflow exists in DATA-Athena-Snowflake — this needs to be built.
- Several analytics Snowflake tables (`ATHIA_STAGE_OUTCOMES`, `ATHIA_SESSION_SUMMARY`) are designed but not deployed. This affects multi-stage monitoring.
- Open questions added: live model status, `ATHIA_` table status in Deuna Snowflake, payment volume for A/B test sizing.
- Pablo granted Rakesh code repository access (2026-02-19 morning).

**Follow-up items to iterate on:** -  Are `ATHIA_PREDICTIONS` / `ATHIA_FEEDBACK` tables populated in Deuna's Snowflake today, or only in Athia's internal environment? — **Confirmed live in Deuna's Snowflake** (2026-02-24) -  Are there live SageMaker endpoints behind `processor_selector` / `retry_predictor` today, or are they placeholders? (Rakesh to confirm) -  What is the current payment volume through the routing engine? Minimum 1,000 transactions per variant needed for A/B test statistical validity. (Ask Israel) -  Who owns athena-platform Go repo deployment — Aidaptive or Deuna infra? This affects Phase 1 deployment planning. (Clarify with Pablo) -  Deploy `ATHIA_STAGE_OUTCOMES` and `ATHIA_SESSION_SUMMARY` tables in Snowflake — needed for multi-stage funnel monitoring (G-05). -  Build `retry_optimization_requested` stimulus in DATA-Athena-Snowflake — no retry-specific LLM workflow exists today (P-05 gap). -  Finish Strategy Director matcher + ranker nodes in DATA-Athena-Snowflake — currently has placeholder code (P-02 gap). -  Confirm whether auto-winner worker is deployed in production with `DRY_RUN=false`, or still in dry-run mode.

## 2026-02-18

- Project plan file created. Details to be filled in.
- Israel is the main POC for data and related topics.
- Pablo is the CTO.
- All data is in Snowflake database; we will get read access to all tables. Snowflake URL: `VLTAXPW-RMONTES.snowflakecomputi`
- Need Claude access and budget for LLM. Farhan is the main POC; Pablo will be talking to Farhan to get this access.
- Mark Walick is the PM lead for this project.

---

## Project Plan Exports

Date	File	Notes
2026-02-18	project-plan-2026-02-18.pdf	Initial export
2026-02-18	project-plan-2026-02-18-v2.pdf	Updated with schema notes, stakeholders, todos
2026-02-18	project-plan-2026-02-18-v3.pdf	Updated with TEAMS.md reference, Mark Walick correction
2026-02-18	project-plan-2026-02-18-v4.pdf	Self-contained: includes project plan + teams + schema
2026-02-18	project-plan-2026-02-18-v5.pdf	Updated project purpose to reflect scoping nature
2026-02-18	project-plan-2026-02-18-v6.pdf	Improved table formatting — fixed column overlaps
2026-02-18	project-plan-2026-02-18-v7.pdf	Latest snapshot
2026-02-18	project-plan-2026-02-18-v8.pdf	Refocused Phase 0 as assessment-only with clear deliverables
2026-02-18	project-plan-2026-02-18-v9.pdf	Added Next Steps section
2026-02-18	schema-2026-02-18.pdf	Initial Snowflake schema snapshot

Date	File	Notes
2026-03-16	project-plan-2026-03-16-v20.pdf	ML platform services + v0.15.5 sync; G-03 partial, G-05/G-13 nearly done; ~75.5d saved, ~29d remaining; Volaris Phase 0 tasks 10/11 [-]
2026-02-26	project-plan-2026-02-26-v19.pdf	Full delivery SOW added (54 tasks, \$120,480, 7 phases, owner assignments); portal updated (effort ~43d remaining, ~61.5d saved); architecture diagrams; Volaris Owner column
2026-02-26	project-plan-2026-02-26-v18.pdf	Full repo sync (both repos); Triton IS branch analysis; G-06 nearly done, G-14 partial; ~60d saved (~45d remaining); Volaris Phase 6 effort updated
2026-02-20	project-plan-2026-02-20-v17.pdf	Updated TRAINING_PLATFORM_TASKS.md with task-level branch status; revised effort estimates (~48d saved)
2026-02-20	project-plan-2026-02-20-v16.pdf	Updated with feat/ATH-0000 branch analysis, gap status updates, new SQL tables
2026-02-19	project-plan-2026-02-19-v12.pdf	Latest export
2026-02-19	project-plan-2026-02-19-v11.pdf	Access status updates, Volaris decision, follow-up items
2026-02-19	project-plan-2026-02-19-v10.pdf	Added full repo analysis: DATA-Athena-Snowflake + athena-platform findings, updated open questions

---

## Documents & SOW Snapshots

Document	Date	Version	File
SOW: Athia Embedded into Acceptance - Phase 1	2026-02-16	v1	PDF

---

## References & Links

- CLAUDE.md (project conventions)
- Deuna Code Repository (GitHub Org)
- Snowflake Data Repository
- Platform Repository
- Data Dictionary
- Athia Data Model
- Snowflake Login (VLTAXPW-RMONTES.snowflakecomputing.com)

---

## Teams & Stakeholders

Source of truth for all people involved in the Smartrouter Project. **Last Updated:** 2026-03-03

## Deuna

---

Name	Role	Responsibilities
Reks	CEO & Co-Founder	Executive leadership
Chema	Co-Founder	Co-founder
Pablo	CTO	Executive sponsor; coordinating Claude/LLM access via Farhan
Israel	Data POC	Main point of contact for data and Snowflake access
Farhan	Claude/LLM Access POC	Provisioning Claude access and budget
Mark Walick	PM Lead	Product management lead

---

## Aidaptive

---

Name	Role	Responsibilities
Rakesh	CEO	Project lead, strategy, Snowflake access verified
Naoki	Solutions Architect	Overall architecture; Snowflake access pending
Rene	ML Engineer	ML model development and training pipeline
Kedar	Backend / Data Engineer	Backend engineering and data infrastructure

---

## Key Contacts by Topic

---

Topic	Owner	Notes
Data / Snowflake	Israel (Deuna)	All data questions, schema, access
Claude / LLM Budget	Farhan (Deuna)	Pablo coordinating with Farhan
Project Management	Mark Walick (Deuna)	
Engineering	Rakesh + Naoki (Aidaptive)	Coordinate with each other on access/setup
ML Engineering	Rene (Aidaptive)	ML model development
Data Engineering	Kedar (Aidaptive)	Data pipeline and backend
Executive Decisions	Pablo (Deuna)	CTO sign-off

---

## Snowflake Schema Reference

Database: PAYMENT\_ML Instance: VLTAXPW-RMONTES.snowflakecomputing.com Extracted: 2026-02-18

---

## Overview

---

Schema	Type	Object	Columns
ABTESTING	Table	ALL_VIEWS_FLAT	~319 (denormalized flat table)
ABTESTING	Table	ALL_VIEWS_FLAT_SAMPLE	~319 (sample of above)
SOURCES	View	VW_ATHENA_CHANNEL	2

---

Schema	Type	Object	Columns
SOURCES	View	VW_ATHENA_ORDER	85
SOURCES	View	VW_ATHENA_ORDER_COMPLEMENT	11
SOURCES	View	VW_ATHENA_PAYMENT	46
SOURCES	View	VW_ATHENA_PAYMENT_ATTEMPT	39
SOURCES	View	VW_ATHENA_PAYMENT_EVENTS	28
SOURCES	View	VW_ATHENA_TARGET_USER	40
SOURCES	View	VW_ORDER_AIRLINE_DETAIL_ALL	29
SOURCES	View	VW_ORDER_AIRLINE_INFORMATION_DETAIL_ALL	51
SOURCES	View	VW_ROUTING_MERCHANT_RULE	14
SOURCES	View	VW_ROUTING_MERCHANT_RULE_CONDITION	16
SOURCES	View	VW_ROUTING_MERCHANT_RULE_MEMBER	15
SOURCES	View	VW_ROUTING_MERCHANT_RULE_OPTION	8
SOURCES	View	VW_ROUTING_MERCHANT_RULE_OPTION_VALUES	8
SOURCES	View	VW_SMART_ROUTING_ATTEMPTS	40

## Schema: ABTESTING

Denormalized flat tables joining all Athena views — used for A/B testing analysis.

### ALL\_VIEWS\_FLAT / ALL\_VIEWS\_FLAT\_SAMPLE

Both tables share the same ~319 columns. ALL\_VIEWS\_FLAT\_SAMPLE is a sampled subset.

Key column groups:

Group	Columns
Identity	SOURCE_TABLE_NAME, CHANNEL_ID, CHANNEL_NAME, COMMERCE_ID, TARGET_USER_ID, USER_ACCOUNT_ID
Order	ORDER_ID, ORDER_DATE, ORDER_TIME, ORDER_STATUS, ORDER_TOKEN, COMMERCE_STORE_CODE
Order Indicators	ORDER_APPROVED_INDICATOR, ORDER_REJECTED_INDICATOR, ORDER_SEND_TO_SMART_ROUTING_INDICATOR, ORDER_RECOVERED_BY_SMART_ROUTING_INDICATOR, ORDER_APPROVED_BY_FIRST_PROCESSOR_INDICATOR, ORDER_DENIED_BY_FRAUD_INDICATOR, ORDER_DENIED_BY_PROCESSOR_INDICATOR
Order Amounts	ORDER_ORIGINAL_GMV_AMOUNT, ORDER_GMV_AMOUNT_USD, ORDER_AUTH_AMOUNT_USD, ORDER_CAPTURE_AMOUNT_USD, ORDER_TOTAL_AMOUNT_USD
Payment	PAYMENT_ID, PAYMENT_DATE, PAYMENT_STATUS, PROCESSOR_NAME, PAYMENT_AMOUNT_USD
Payment Attempt	PAYMENT_ATTEMPT_ID, PAYMENT_ATTEMPT_SEQUENCE_ORDER, PAYMENT_ATTEMPT_STATUS, PAYMENT_ATTEMPT_PROCESSOR_NAME, PAYMENT_ATTEMPT_ERROR_CODE, PAYMENT_ATTEMPT_APPROVED_INDICATOR
Event	EVENT_TYPE, EVENT_STATUS, EVENT_CREATED_AT, EVENT_ERROR_CODE, EVENT_ERROR_STANDARD_ERROR_CODE
Card	CARD_BIN, CARD_BRAND, CARD_LAST_FOUR, CARD_COUNTRY, BANK

Group	Columns
Fraud	FRAUD_PROCESSOR_NAME, FRAUD_RISK_LEVEL, FRAUD_RISK_SCORE, FRAUD_STATUS
User	TARGET_USER_BROWSER, TARGET_USER_OS, TARGET_USER_DEVICE, TARGET_USER_FRAUD_RATE_COHORT, TARGET_USER_TENURE_IN_DAYS
Routing Rules	RULE_ID, PROPERTIES__RULES_LABEL, MERCHANT_PAYMENT_PROCESSOR_NAME, COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID
Geo	LATITUDE, LONGITUDE, ORDER_CITY_NAME, ORDER_STATE_NAME, ORDER_COUNTRY_CODE, WEATHER_MAIN
Airline	PNR, FLIGHT_NUMBER, CARRIER_CODE, DESTINATION_IATA_CODE, TOTAL_PASSENGER

## Schema: SOURCES

Raw source views feeding the ABTESTING schema. Join key across most views: COMMERCE\_ID, ORDER\_ID, PAYMENT\_ID, PAYMENT\_ATTEMPT\_ID.

### VW\_ATHENA\_CHANNEL (2 cols)

Channel lookup table.

Column	Type
CHANNEL_ID	NUMBER(5,0)
CHANNEL_NAME	VARCHAR

### VW\_ATHENA\_ORDER (85 cols)

Core order-level data including status, amounts, payment method, behavioral signals, and geo.

Column	Type	Notes
COMMERCE_ID	VARCHAR	Merchant ID
TARGET_USER_ID	VARCHAR(32)	User ID
USER_ACCOUNT_ID	VARCHAR(32)	
CHANNEL_ID	NUMBER	
ORDER_ID	VARCHAR	Primary key
ORDER_DATE / ORDER_TIME	DATE / TIME	
ORDER_STATUS	VARCHAR	
ORDER_APPROVED_INDICATOR	BOOLEAN	
ORDER_SEND_TO_SMART_ROUTING_INDICATOR	BOOLEAN	Was smart routing used?
ORDER_RECOVERED_BY_SMART_ROUTING_INDICATOR	BOOLEAN	Did smart routing recover?
ORDER_DENIED_BY_FRAUD_INDICATOR	BOOLEAN	
ORDER_ORIGINAL_GMV_AMOUNT / _USD	FLOAT	
ORDER_AUTH_AMOUNT_USD	FLOAT	

Column	Type	Notes
ORDER_TOTAL_AMOUNT_USD	FLOAT	
PAYMENT_CURRENCY	VARCHAR	
CARD_LAST_FOUR / CARD_COUNTRY	VARCHAR	
DEVICEID / REQUEST_IP	VARCHAR	
USER_IS_GUEST	BOOLEAN	
TOTA_MINUTES_BROWSING	NUMBER	Behavioral feature
TOTAL_EVENTS_BEFORE_PURCHASE	NUMBER	Behavioral feature
TOTAL_NUM_SESSIONS	NUMBER	Behavioral feature
LATITUDE / LONGITUDE	NUMBER	
WEATHER_MAIN	VARCHAR	
ORDER_TOKEN	VARCHAR(100)	

### VW\_ATHENA\_ORDER\_COMPLEMENT (11 cols)

Fraud and 3DS signals at the order level.

Column	Type
COMMERCE_ID	VARCHAR
CHANNEL_ID	NUMBER
ORDER_ID	VARCHAR
FRAUD_PROCESSOR_NAME	VARCHAR
FRAUD_RISK_LEVEL	VARCHAR
FRAUD_RISK_SCORE	FLOAT
FRAUD_STATUS	VARCHAR
SITEDOMAIN	VARCHAR
WEBSITENAME	VARCHAR
CHALLENGE_3DS_INDICATOR	BOOLEAN
CHALLENGE_3DS_STATUS	VARCHAR

### VW\_ATHENA\_PAYMENT (46 cols)

Payment-level data: processor, card info, error codes, routing rules.

Column	Type	Notes
PAYMENT_ID	VARCHAR(250)	Primary key
ORDER_ID	VARCHAR	FK → Order
PAYMENT_DATE / PATMENT_TIME	DATE / TIME	Note: typo in source (PATMENT)
PAYMENT_STATUS	VARCHAR	
PROCESSOR_NAME	VARCHAR	
CARD_BIN / CARD_BRAND / BANK	VARCHAR	
NUM_ATTEMPTS_ORDER	NUMBER	
NUM_ATTEMPTS_SMART_ROUTING	NUMBER	
ERROR_MESSAGE / ERROR_CODE / ERROR_CATEGORY	VARCHAR	

Column	Type	Notes
PAYMENT_AMOUNT_USD	FLOAT	
HARD_SOFT	VARCHAR	Hard vs soft decline
RULE_ID	VARCHAR	Routing rule applied
PROPERTIES__RULES_LABEL	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_NAME	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_ID	VARCHAR	
PREVIOUS_ORDER_ERROR_CODE	VARCHAR	Prior attempt context
PREVIOUS_ORDER_PROCESSOR	VARCHAR	
AUTHORIZATION_CODE	VARCHAR	
COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID	VARCHAR(36)	

### VW\_ATHENA\_PAYMENT\_ATTEMPT (39 cols)

Individual attempt-level data — key table for retry optimization.

Column	Type	Notes
PAYMENT_ATTEMPT_ID	VARCHAR(32)	Primary key
PAYMENT_ID	VARCHAR(250)	FK → Payment
ORDER_ID	VARCHAR	FK → Order
PAYMENT_ATTEMPT_SEQUENCE_ORDER	NUMBER	Attempt number
PAYMENT_LAST_ATTEMPT_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_STATUS	VARCHAR	
PAYMENT_ATTEMPT_PROCESSOR_NAME	VARCHAR	Which processor used
PAYMENT_ATTEMPT_PROCESSOR_CODE	VARCHAR	
PAYMENT_ATTEMPT_ERROR_CODE	VARCHAR	
PAYMENT_ATTEMPT_ERROR_CATEGORY	VARCHAR	
PAYMENT_ATTEMPT_HARD_SOFT_TYPE	VARCHAR	
PAYMENT_ATTEMPT_RETRY_INDICATOR	VARCHAR	
PAYMENT_ATTEMPT_APPROVED_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_ACCEPTANCE_RATE_INDICATOR	BOOLEAN	
PAYMENT_ATTEMPT_AMOUNT_USD	FLOAT	
PAYMENT_ATTEMPT_CARD_BRAND / CARD_BIN / BANK	VARCHAR	
DENIED_BY_PSP_OR_FRAUD	VARCHAR	
DYNAMIC_ROUTING_DETAIL	VARIANT	JSON routing detail
RULE_ID	VARCHAR	
MERCHANT_PAYMENT_PROCESSOR_ID	VARCHAR	
COMMERCE_ROUTING_MERCHANT_RULE_VERSION_ID	VARCHAR(36)	

### VW\_ATHENA\_PAYMENT\_EVENTS (28 cols)

Event stream for each payment attempt — captures state transitions.

Column	Type	Notes
PAYMENT_ATTEMPT_ID	VARCHAR(32)	FK → Attempt
PAYMENT_ATTEMP_EVENT_INDEX	NUMBER	Event order within attempt
EVENT_TYPE	VARCHAR	
EVENT_STATUS	VARCHAR	
EVENT_CREATED_AT	TIMESTAMP_NTZ	
EVENT_ORIGINAL_TOTAL_AMOUNT	NUMBER	
EVENT_ERROR_CODE	VARCHAR	
EVENT_ERROR_STANDARD_ERROR_CODE	VARCHAR	Normalized error code
EVENT_ERROR_STANDARD_ERROR_MESSAGE	VARCHAR	
EVENT_ERROR_DEUNA	VARCHAR	Deuna-specific error
EVENT_REFUND_VOID_REASON	VARCHAR	

### VW\_ATHENA\_TARGET\_USER (40 cols)

User profile and behavioral signals.

Column	Type	Notes
TARGET_USER_ID	VARCHAR(32)	Primary key
COMMERCE_ID	VARCHAR	
TARGET_USER_BROWSER / OS / DEVICE / EQUIPMENT	VARCHAR	Device fingerprint
TARGET_USER_FAVORITE_PAYMENT_METHOD	VARCHAR	
TARGET_USER_FAVORITE_CARD_BRAND / BANK	VARCHAR	
TARGET_USER_ACCESS_COUNTRY_CODE	VARCHAR	
TARGET_USER_FIRST_PURCHASE_DATE	TIMESTAMP	
TARGET_USER_LAST_PURCHASE_DATE	TIMESTAMP	
TARGET_USER_USER_FRAUD_RATE	NUMBER	
TARGET_USER_FRAUD_RATE_COHORT	VARCHAR(30)	
TARGET_USER_TENURE_IN_DAYS	NUMBER	
TARGET_USER_FREQUENCY_VALUE	NUMBER	RFM frequency
TARGET_USER_RECENCY_VALUE	NUMBER	RFM recency
TARGET_USER_MONETARY_VALUE	FLOAT	RFM monetary
TARGET_USER_NUM_ORDERS_VALUE	NUMBER	

### VW\_ORDER\_AIRLINE\_DETAIL\_ALL (29 cols)

Airline booking details (Volaris-specific). Joined via ORDER\_ID.

Key fields: PNR, BOOKINGISINTERNATIONAL, NAVITAIRE\_CARRIER\_CODE, TOTAL\_FLIGHT\_NUMBERS, TOTAL\_PASSENGER, ROUND\_FLIGHT\_IND

### VW\_ORDER\_AIRLINE\_INFORMATION\_DETAIL\_ALL (51 cols)

Flight + passenger details per order. Joined via ORDER\_ID.

Key fields: FLIGHT\_NUMBER, CARRIER\_CODE, ORIGIN\_IATA\_CODE, DESTINATION\_IATA\_CODE, PASSENGER\_TYPE, PASSENGER\_FREQUENT\_FLYER\_CODE, SERVICE\_CLASS, TOTAL\_AMOUNT\_USD

---

### **VW\_ROUTING\_MERCHANT\_RULE (14 cols)**

Merchant routing rules configuration.

---

Column	Type
ID	NUMBER
MERCHANT_ID	VARCHAR
LABEL	VARCHAR
STATUS	VARCHAR
PRIORITY	NUMBER
TRIGGER_	VARCHAR
IS_DEFAULT	VARCHAR
IGNORE_NEXT_RULES	VARCHAR
MERCHANT_RULE_PARENT	NUMBER
CREATED_AT / UPDATED_AT / DELETED_AT	TIMESTAMP

---

---

### **VW\_ROUTING\_MERCHANT\_RULE\_CONDITION (16 cols)**

Conditions that trigger routing rules.

Key fields: MERCHANT\_RULE\_ID, MERCHANT\_RULE\_OPTION\_ID, OPERAND, OPERAND\_FIELD\_TO\_EVALUATE, OPERATOR, METADATA\_FIELD\_NAME

---

### **VW\_ROUTING\_MERCHANT\_RULE\_MEMBER (15 cols)**

Processors assigned to routing rules.

Key fields: MERCHANT\_RULE\_ID, PAYMENT\_PROCESSOR\_ID, MERCHANT\_PAYMENT\_PROCESSOR\_ID, STRATEGY, SORT, SHADOW\_MODE, CAPABILITIES, FRAUD\_PROCESSOR

---

### **VW\_ROUTING\_MERCHANT\_RULE\_OPTION (8 cols)**

Available routing rule option types.

Key fields: ID, LABEL, OPERATORS\_AVAILABLE

---

### **VW\_ROUTING\_MERCHANT\_RULE\_OPTION\_VALUES (8 cols)**

Allowed values for routing rule options.

Key fields: ID, MERCHANT\_RULE\_OPTION, VALUE\_

---

### **VW\_SMART\_ROUTING\_ATTEMPTS (40 cols)**

Event stream from the smart routing engine — per-attempt routing decisions.

Column	Type	Notes
ATTEMPT_ID	NUMBER	
PROPERTIES_TRANSACTION_ID	VARCHAR	Links to payment
PROPERTIES_MERCHANT_ID	VARCHAR	
PROPERTIES_ALGORITHM_TYPE	VARCHAR	Which routing algorithm
RULE_ID	NUMBER	Rule applied
PROPERTIES_GATEWAY	BOOLEAN	
PROPERTIES_PAYMENT_PROCESSOR_ID	NUMBER	
PROPERTIES_PROCESSOR_CODE	VARCHAR	
PROPERTIES_RESULT_STATUS	VARCHAR	
PROPERTIES_RESULT_ERROR_CODE	VARCHAR	
PROPERTIES_RESULT_PROCESS_TIME	FLOAT	Latency signal
PROPERTIES_RESULT_SKIPPED_REASON	VARCHAR	Why processor was skipped
PROPERTIES_FRANCHISE / COUNTRY / CITY / STATE	VARCHAR	
PROPERTIES_ORDER_VALUE	NUMBER	
ORIGINAL_TIMESTAMP / RECEIVED_AT	TIMESTAMP	

## Key Relationships

### VW\_ATHENA\_CHANNEL

CHANNEL\_ID → VW\_ATHENA\_ORDER

### VW\_ATHENA\_ORDER

ORDER\_ID → VW\_ATHENA\_ORDER\_COMPLEMENT

ORDER\_ID → VW\_ATHENA\_PAYMENT

ORDER\_ID → VW\_ORDER\_AIRLINE\_DETAIL\_ALL

ORDER\_ID → VW\_ORDER\_AIRLINE\_INFORMATION\_DETAIL\_ALL

TARGET\_USER\_ID → VW\_ATHENA\_TARGET\_USER

### VW\_ATHENA\_PAYMENT

PAYMENT\_ID → VW\_ATHENA\_PAYMENT\_ATTEMPT

RULE\_ID → VW\_ROUTING\_MERCHANT\_RULE

### VW\_ATHENA\_PAYMENT\_ATTEMPT

PAYMENT\_ATTEMPT\_ID → VW\_ATHENA\_PAYMENT\_EVENTS

PROPERTIES\_TRANSACTION\_ID → VW\_SMART\_ROUTING\_ATTEMPTS

### VW\_ROUTING\_MERCHANT\_RULE

ID → VW\_ROUTING\_MERCHANT\_RULE\_CONDITION

ID → VW\_ROUTING\_MERCHANT\_RULE\_MEMBER

### ABTESTING.ALL\_VIEWS\_FLAT

Denormalized join of all above views