ArqAI FWA Detection POC: Updated Technical Architecture

Incorporating Domain Expert Requirements from Sunil Pal

Last Updated: October 31, 2025
Based on: Technical review call + FWA rules specification

Critical Architecture Changes

1. Rule Engine Redesign (Major Change)

Old Approach: Hard-coded fraud detection rules
New Approach: Configurable rule engine with admin UI

What Changed:

- **29 total rules** must be configurable:
    - 16 Medical FWA rules (M1-M20)
    - 13 Pharmacy FWA rules (P1-P15)
- Admin users can adjust thresholds without code changes
- Rules stored as policy-as-code with version control

Example Rule Configuration:

```
{
  "rule_id": "M1",
  "category": "Upcoding",
  "fraud_type": "Fraud",
  "detection_logic": {
    "condition": "CPT_billed > expected_CPT_for_diagnosis",
    "threshold_percent": 20,  // Admin-configurable
    "threshold_amount": 300,  // Admin-configurable
    "benchmark_source": "CMS_fee_schedule"
  },
  "severity_weight": 8.5,
  "enabled": true,
```

"last_modified": "2025-10-31",

"modified_by": "admin@healthplan.com"

}

2. Simplified Policy Enforcement (Removed Components)

Removed from Architecture:

- ❌ HIPAA Access Controls box (moved to infrastructure layer)
- ❌ Anti-Discrimination check (embedded in rule engine as gender/age appropriateness)

Why:

- HIPAA = data vault/anonymization at infrastructure level, not fraud detection
- Anti-discrimination = clinical appropriateness rules (e.g., pregnancy test to male = fraud flag)

Updated Flow:

Risk Scoring → Policy Engine → CMS Compliance Guidelines Check → Pass/Fail

3. Manual Review & Resolution Workflow (Critical Addition)

Problem Identified: Original architecture stopped after flagging fraud—no workflow for case resolution.

New Post-Investigation Engine:

Investigation Queue → AI Agent Communication (optional)

↓

Case Resolution Paths:

Path 1: Provider Accepts → Claim Corrected → Case Closed

Path 2: Provider Disputes → Rule Validity Review → Update Rule Engine

Path 3: Plan Benefit Issue → Escalate to Plan Admin → Benefit Configuration

Path 4: No Response After N Attempts → Manual Triage → Ticketing System

Path 5: Complex Case → Manual Investigation → Recovery Process

Implementation Requirements:

- Ticketing system integration (ServiceNow, Jira, custom)

- Email/API communication with claim processors
- Case status tracking (Open → Under Review → Resolved → Closed)
- SLA monitoring (aging reports for unresolved cases)

4. Data Sources Architecture (External Dependencies)

Client Provides:

- Medical claims (CPT/HCPCS codes, ICD-10 diagnosis, amounts, dates)
- Pharmacy claims (NDC codes, fill dates, quantities, prescriber IDs)
- Member eligibility data
- Provider/pharmacy directories (basic info)

ArqAI Must Source Externally:

Data Element	Source	Update Frequency	Purpose
NDC Reference	CMS National Drug Code Directory	Monthly	Drug name, class, cost benchmark
CPT/HCPCS Codes	CMS Physician Fee Schedule	Quarterly	Procedure definitions, expected costs
ICD-10 Diagnosis	CMS ICD-10 Database	Annual	Diagnosis definitions for upcoding detection
Provider NPI Registry	NPPES (National Plan Provider Enumeration System)	Weekly	Active provider validation
DEA Registration	DEA Diversion Control	Monthly	Prescriber authorization for controlled substances
OIG Exclusions	OIG LEIE (List of Excluded Individuals/Entities)	Monthly	Sanctioned provider detection

Medicare/Medicaid Benchmarks	CMS Fee Schedules	Quarterly	Cost comparison for overbilling
Clinical Guidelines	CMS LCD/NCD (Local/National Coverage Determinations)	Quarterly	Medical necessity validation

API Integration Strategy:

```
# Example: Real-time NPI validation
def validate_provider_npi(npi):
  """Check if provider NPI is active and not sanctioned"""

  # Check 1: NPPES active status
  nppes_response = requests.get(
    f"https://npiregistry.cms.hhs.gov/api/?number={npi}&version=2.1"
  )
  if not nppes_response.json()['result_count']:
    return {"valid": False, "reason": "NPI not found in NPPES"}

  # Check 2: OIG exclusion list
  oig_response = requests.get(
    f"https://oig.hhs.gov/exclusions/exclusions_list.asp?npi={npi}"
  )
  if oig_response.status_code == 200:
    return {"valid": False, "reason": "Provider on OIG exclusion list"}

  # Check 3: DEA registration for prescribers
  if is_prescriber(npi):
    dea_valid = check_dea_registration(npi)
    if not dea_valid:
      return {"valid": False, "reason": "DEA registration expired/invalid"}

  return {"valid": True, "npi_details": nppes_response.json()}
```

5. Risk Scoring Algorithm (Needs Definition)

Question from Sunil: "How are you scoring it? What is the definition of scoring?"

Proposed Approach (To Be Validated):

Base Scoring Model:

Total Risk Score (0-100) = Weighted Sum of Triggered Rules

Each rule contributes: Rule_Weight × Severity_Multiplier × Confidence_Factor

Where:

- Rule_Weight: Pre-defined importance (Medical upcoding = 8.5, Pharmacy early refill = 3.2)

- Severity_Multiplier: How much rule was violated (200% over benchmark vs 20% over)

- Confidence_Factor: Data quality and completeness (missing fields reduce confidence)

Example Calculation:

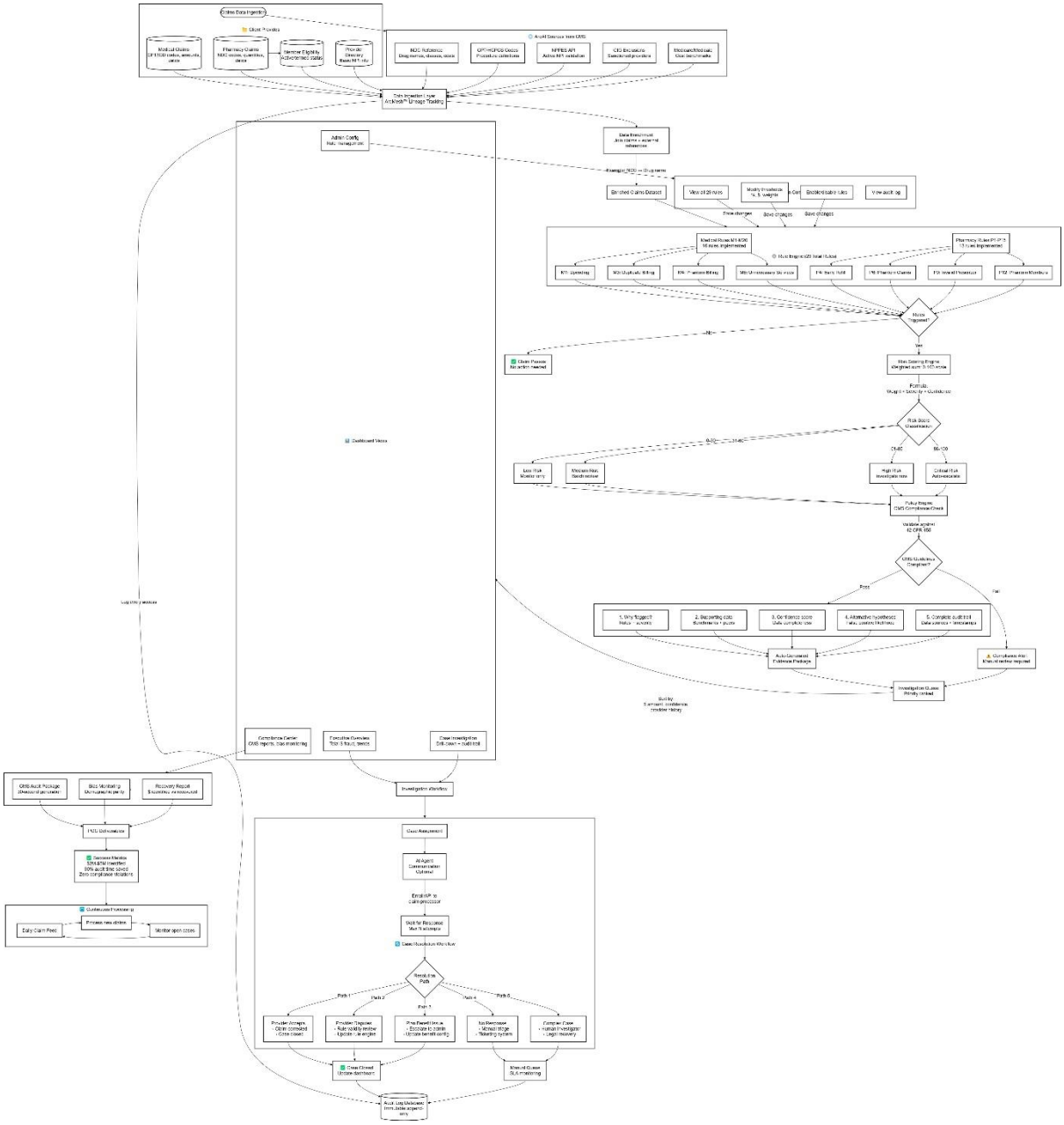
```
def calculate_fraud_risk_score(claim_id, triggered_rules):
  """
  Calculate 0-100 risk score for a claim
  """
  base_score = 0
  max_possible_score = 0

  for rule in triggered_rules:
    # Rule M1: Upcoding
    if rule['rule_id'] == 'M1':
      weight = 8.5
      billed_amount = claim['amount_billed']
      expected_amount = get_cms_benchmark(claim['cpt_code'])

      # Severity: How much over expected?
      overpayment_percent = (billed_amount - expected_amount) / expected_amount
```

```
        severity_multiplier = min(overpayment_percent / 0.20, 3.0)  # Cap at 3x

        # Confidence: Do we have all required data?
        confidence = 1.0
        if not claim['diagnosis_code']:
            confidence *= 0.7
        if not claim['length_of_stay']:
            confidence *= 0.8

        rule_contribution = weight * severity_multiplier * confidence
        base_score += rule_contribution
        max_possible_score += weight * 3.0  # Maximum possible

    # Normalize to 0-100 scale
    normalized_score = (base_score / max_possible_score) * 100 if max_possible_score > 0 else 0

    return min(normalized_score, 100)
```

Thresholds (Admin-Configurable):

- **0-30:** Low Risk (automated approval, monitoring only)
- **31-60:** Medium Risk (secondary review, batched investigation)
- **61-85:** High Risk (immediate investigation queue)
- **86-100:** Critical Risk (automatic escalation to fraud investigator)

Updated System Architecture



Separate HTML file attached with email for better visibility

Technical Implementation Priorities

Phase 1: Core Detection (Weeks 1-2)

1. **Data ingestion pipeline**
    - Client claims data (CSV/API)
    - External CMS data sources (NDC, CPT, NPI)
    - Data enrichment and validation

2. **Rule engine foundation**
    - Implement 5 highest-priority rules first:
        - M1: Upcoding (Medical)
        - M3: Duplicate Billing (Medical)
        - P4: Early Refill (Pharmacy)
        - P6: Phantom Claims (Pharmacy)
        - M4: Phantom Billing (Medical)
3. **Basic risk scoring**
    - Simple weighted sum algorithm
    - Threshold classification

Phase 2: Governance Layer (Week 3)

4. **ArqAI IP implementation**
    - ArqMesh™ lineage tracking (audit logs)
    - Compliance-aware policy validation
    - Explainability module (evidence generation)
5. **Admin dashboard MVP**
    - View existing rules
    - Adjust thresholds (no add/delete rules yet)

Phase 3: Investigation Workflow (Week 4)

6. **Manual review workflow**
    - Investigation queue with prioritization
    - Case assignment and status tracking
    - Basic ticketing system integration
7. **Dashboard completion**
    - Executive overview
    - Case investigation workspace
    - Compliance reporting

Phase 4: Advanced Features (Post-POC)

8. **AI agent communication** (optional for POC)
9. **Full rule engine** (all 29 rules)
10. **Advanced analytics** (provider network graphs, predictive models)

Data Sources: Integration Details

Priority 1: Essential for POC

CMS National Drug Code Directory

- URL: https://www.fda.gov/drugs/drug-approvals-and-databases/national-drug-code-directory
- Format: CSV download (monthly updates)
- Fields needed: NDC, Proprietary Name, Non-Proprietary Name, Dosage Form, Route, DEA Schedule
- Integration: Monthly batch download, load into reference database

NPPES NPI Registry

- API: https://npiregistry.cms.hhs.gov/api/
- Real-time lookup endpoint
- Example: GET /api/?version=2.1&number={NPI}
- Rate limit: 1,200 requests/min
- Fields: Provider name, specialty, practice location, status

OIG Exclusions List

- URL: https://oig.hhs.gov/exclusions/exclusions_list.asp
- Format: CSV download
- Update frequency: Monthly
- Integration: Download and validate against provider NPI

CMS Physician Fee Schedule

- URL: https://www.cms.gov/medicare/physician-fee-schedule/search
- Format: Downloadable database (quarterly)
- Fields: CPT/HCPCS code, description, facility/non-facility price
- Integration: Quarterly refresh of cost benchmarks

Priority 2: Enhanced Detection (Post-POC)

DEA Diversion Control

- Prescriber validation
- Controlled substance monitoring

Clinical Guidelines (LCD/NCD)

- Medical necessity validation
- Procedure appropriateness checks

Risk Scoring: Detailed Specification

Rule Weight Matrix (To Be Validated with Sunil)

Critical Rules (Weight: 8-10):

- M1: Upcoding (9.0)
- M4: Phantom Billing (10.0)
- P6: Phantom Claims (10.0)
- M5: Kickback/Self-Referral (9.5)
- P5: Controlled Substance Diversion (9.5)

High-Impact Rules (Weight: 6-8):

- M2: Unbundling (7.5)
- M6: Medically Unnecessary Services (7.0)
- P2: Doctor Shopping (7.5)
- P12: Phantom Members (8.0)

Medium-Impact Rules (Weight: 4-6):

- M8: Modifier Misuse (5.5)
- P4: Early Refill (4.5)
- M12: Lab/Diagnostic Abuse (5.0)
- P10: Stockpiling (4.0)

Low-Impact Rules (Weight: 1-3):

- M9: Copay Waiver (2.5)

- P3: Pharmacy Shopping (3.0)
- M17: Excessive Procedures (3.5)

Severity Multiplier Calculation

```
def calculate_severity_multiplier(rule_id, claim_data, benchmark_data):
    """
    Calculate how severely a rule was violated
    Returns: 0.1 to 3.0 multiplier
    """

    if rule_id == 'M1':  # Upcoding
        billed = claim_data['amount_billed']
        expected = benchmark_data['expected_amount']

        overpayment_ratio = (billed - expected) / expected

        # Graduated severity
        if overpayment_ratio < 0.10:
            return 0.5  # Minor overage (10%)
        elif overpayment_ratio < 0.25:
            return 1.0  # Moderate (10-25%)
        elif overpayment_ratio < 0.50:
            return 1.8  # Significant (25-50%)
        else:
            return 3.0  # Severe (>50%)

    elif rule_id == 'P4':  # Early Refill
        days_supply = claim_data['days_supply']
        days_since_last_fill = claim_data['days_since_last_fill']

        expected_refill_day = days_supply * 0.75
        early_by_days = expected_refill_day - days_since_last_fill
```

```
    if early_by_days < 3:
        return 0.3  # Borderline early
    elif early_by_days < 7:
        return 0.8  # Somewhat early
    elif early_by_days < 14:
        return 1.5  # Significantly early
    else:
        return 2.5  # Extremely early (stockpiling)

# Default for boolean rules (triggered or not)
return 1.0
```

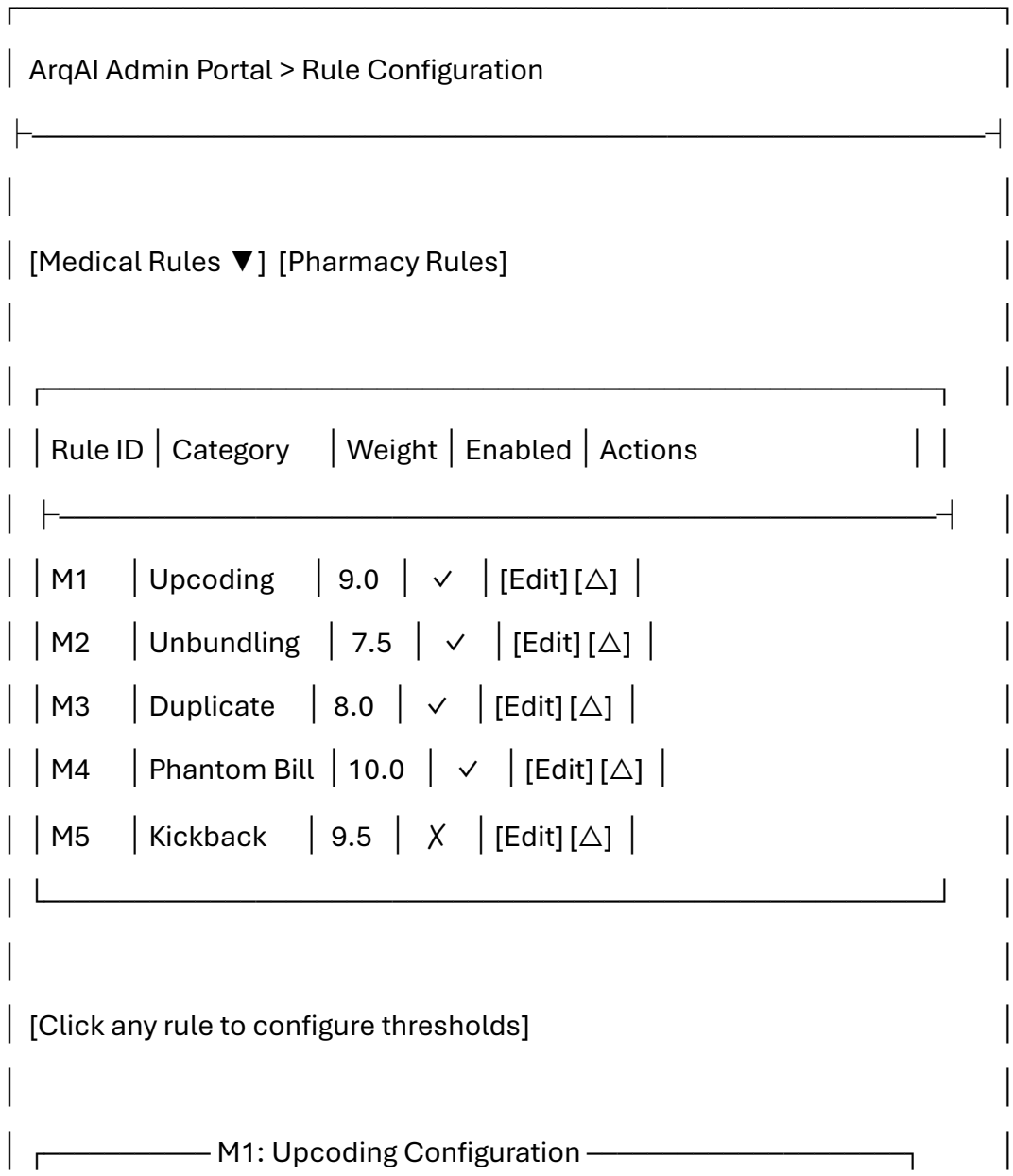
Confidence Factor

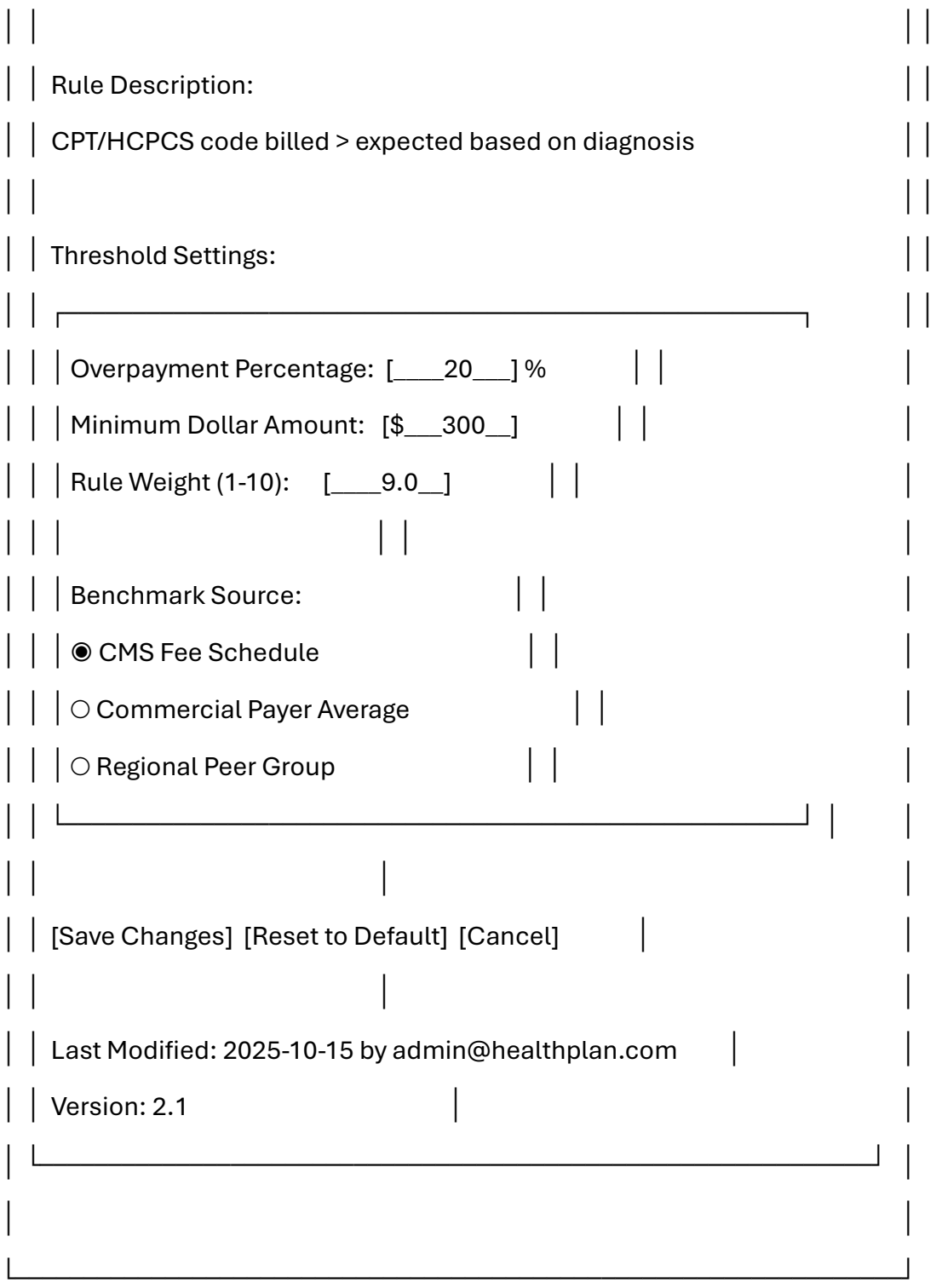
```
def calculate_confidence_factor(claim_data):
    """
    Adjust score based on data completeness and quality
    Returns: 0.3 to 1.0
    """
    confidence = 1.0

    # Penalize missing data
    if not claim_data.get('diagnosis_code'):
        confidence *= 0.7
    if not claim_data.get('procedure_code'):
        confidence *= 0.6
    if not claim_data.get('provider_specialty'):
        confidence *= 0.8
    if not claim_data.get('length_of_stay'):
        confidence *= 0.9

    # Boost for strong corroborating evidence
```

```
    if claim_data.get('ehr_encounter_verified'):
        confidence *= 1.1
    if claim_data.get('multiple_rules_triggered'):
        confidence *= 1.15

    return max(0.3, min(confidence, 1.0))  # Clamp between 0.3-1.0
```

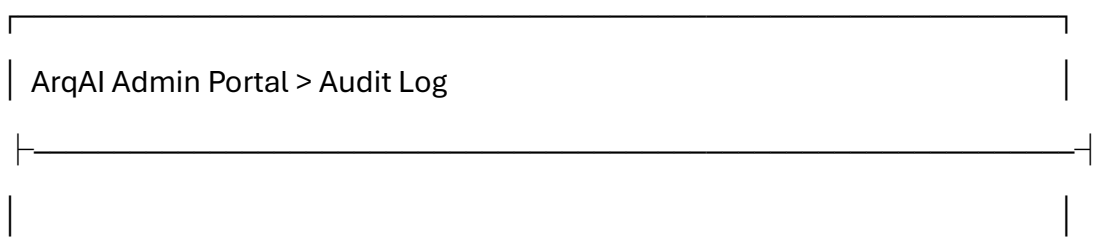
Admin Configuration UI (Wireframe Specification)

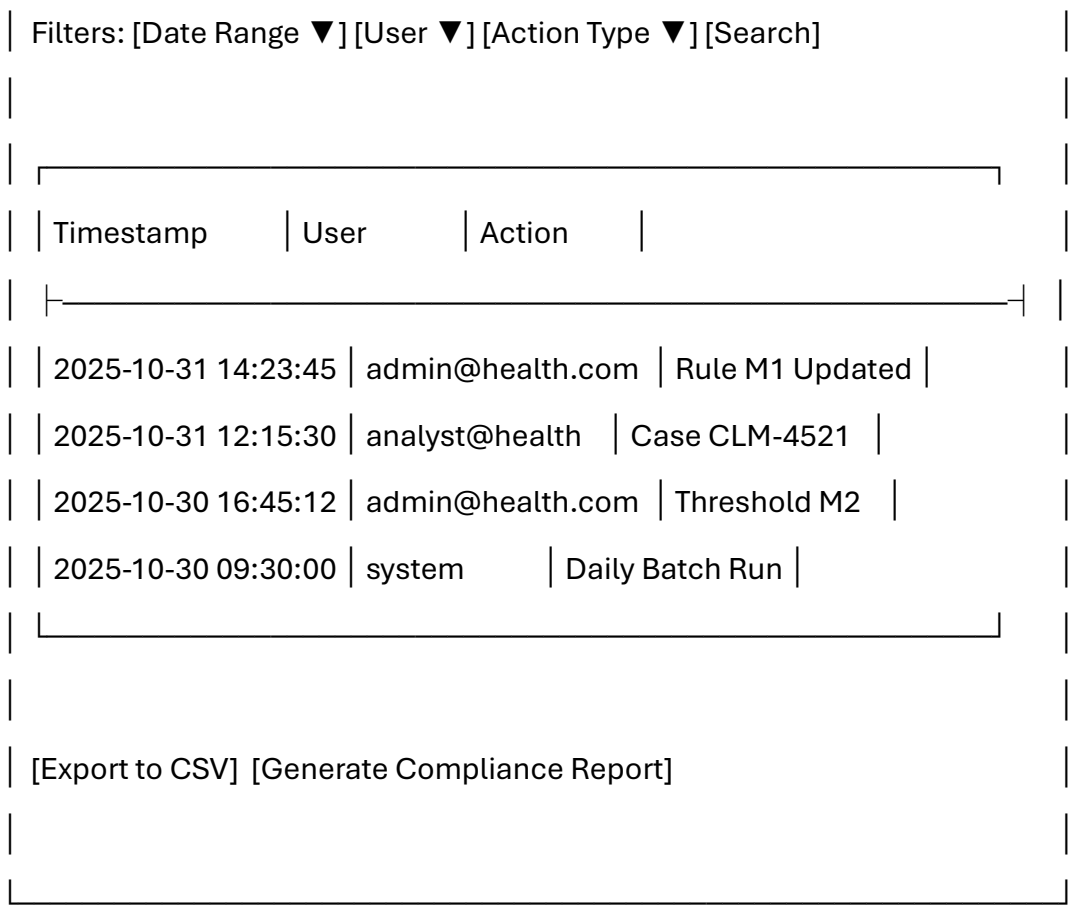
Rule Management Screen





Audit Log Viewer





Implementation Roadmap: Updated Based on Sunil's Feedback

Week 1: Foundation & External Data Integration

Day 1-2: Data Pipeline Setup

- Set up data ingestion for client claims (medical + pharmacy)
- Build data enrichment engine
- Integrate CMS data sources:
    - NPPES NPI Registry API
    - NDC Directory (download and load)
    - CPT/HCPCS fee schedule

Day 3-4: Rule Engine Foundation

- Implement rule storage structure (JSON/database)
- Build rule evaluation framework
- Implement first 5 priority rules:

    - M1: Upcoding
    - M3: Duplicate Billing
    - M4: Phantom Billing
    - P4: Early Refill
    - P6: Phantom Claims

Day 5: Risk Scoring

- Implement scoring algorithm
- Configure rule weights
- Test threshold classification

Deliverable: Can ingest claims + external data, apply 5 rules, generate risk scores

Week 2: ArqAI Governance Layer

Day 6-7: Audit Trail & Lineage

- Implement ArqMesh™ data lineage tracking
- Build audit log database (immutable append-only)
- Log all data access, rule application, decisions

Day 8-9: Explainability Module

- Auto-generate evidence packages
- Natural language explanation templates
- Confidence scoring implementation

Day 10: Policy Validation

- CMS compliance guidelines check
- Pass/fail decision logic

Deliverable: Every fraud detection has complete audit trail + explainability

Week 3: Investigation Workflow

Day 11-12: Investigation Queue

- Build prioritization logic ($ amount, confidence, history)
- Case assignment workflow
- Status tracking (Open → In Progress → Resolved → Closed)

Day 13-14: Manual Review Workflow

- Basic ticketing system integration (or internal tracking)
- Resolution path logic (accept, dispute, escalate)
- SLA monitoring for aging cases

Day 15: Dashboard MVP

- Executive overview page
- Case investigation workspace
- Basic search/filter functionality

Deliverable: End-to-end workflow from detection → investigation → resolution

Week 4: Admin Tools & Polish

Day 16-17: Admin Configuration UI

- Rule management screen
- Threshold adjustment interface
- View/enable/disable rules

Day 18-19: Compliance Dashboard

- CMS audit report generation
- Policy enforcement logs
- Evidence package export (PDF)

Day 20: Testing & Refinement

- End-to-end testing with synthetic data
- Performance optimization
- Bug fixes

Deliverable: Production-ready POC with admin tools

Key Questions to Resolve with Sunil (Next Call)

1. Risk Scoring Algorithm Validation

Question: Is the weighted sum approach acceptable, or do you recommend a different scoring methodology?

Proposed approach:

Risk Score = Σ (Rule_Weight × Severity_Multiplier × Confidence_Factor)

Normalized to 0-100 scale

Alternative approaches to consider:

- Machine learning model (trained on historical fraud cases)
- Bayesian probability scoring
- Simple additive model (each rule = fixed points)

2. Rule Weight Calibration

Question: Do the proposed rule weights align with real-world fraud impact?

Need validation on:

- M1 Upcoding: Weight 9.0 (is this severe enough?)
- M4/P6 Phantom Billing: Weight 10.0 (highest severity justified?)
- P4 Early Refill: Weight 4.5 (too low? Should wasteful patterns be higher?)

Request: Provide example historical cases with known outcomes to calibrate weights

3. External Data Update Frequency

Question: How frequently should we refresh external data sources?

Current plan:

- NPPES NPI: Weekly API calls for active validation
- NDC Directory: Monthly batch download
- CMS Fee Schedule: Quarterly update
- OIG Exclusions: Monthly check

Concern: Real-time validation (NPPES API) may introduce latency. Acceptable trade-off?

4. Manual Review Workflow Complexity

Question: How many resolution paths should POC support?

Minimum viable:

- Path 1: Accept → Close case
- Path 4: No response → Manual queue

Full implementation:

- All 5 paths with AI agent communication

Recommendation for POC: Start with minimum, add complexity post-validation?

5. Admin UI Scope

Question: Should admin UI allow adding new rules, or only modify existing?

Option A (Simple): View + modify thresholds only **Option B (Complex):** View + modify + add/delete rules

POC Recommendation: Option A, defer rule creation to post-POC

6. Data Volume Assumptions

Question: What claim volume should POC architecture support?

Assumptions for mid-sized MA plan (100K members):

- Medical claims: ~1.5M claims/year (125K/month, 4K/day)
- Pharmacy claims: ~2M claims/year (165K/month, 5.5K/day)

Performance targets:

- Batch processing: <30 minutes for daily volume
- Real-time detection: <5 seconds per claim

Validation needed: Are these volumes realistic for target customers?

Success Metrics: Updated for POC

Primary Metrics (Must Achieve)

1. **Fraud Detection Accuracy**
    - Identify $2M-$5M in potential fraud (30-day historical analysis)
    - False positive rate <15% (validated against known legitimate claims)
    - Coverage: At least 10 of 29 rules implemented and operational
2. **Audit Documentation Efficiency**
    - Generate complete CMS audit package in <60 seconds
    - 90% reduction in manual documentation time (3 hours vs 40+ hours)
    - 100% of flagged cases have complete audit trail

3. **Governance Compliance**
    - 100% of detections validated against CMS compliance guidelines
    - Zero policy violations in investigation workflow
    - Complete data lineage for all cases

Secondary Metrics (Nice to Have)

4. **Investigation Workflow**
    - Average case resolution time <5 days
    - 70%+ cases auto-triaged correctly
    - <10% cases require escalation to manual review
5. **User Experience**
    - Admin can configure rule thresholds in <5 minutes
    - Investigators can review case evidence in <2 minutes
    - Dashboard load time <3 seconds
6. **System Performance**
    - Process 10K claims in <30 minutes (batch mode)
    - Real-time detection <5 seconds per claim
    - 99.5% uptime during POC period

Technical Stack: Final Recommendations

Core Infrastructure

Language & Framework:

Backend: Python 3.11+ with FastAPI

Why: Fast development, rich healthcare/ML libraries, async support

Data Processing: Polars (primary) + pandas (compatibility)

Why: Polars 10x faster for large claims datasets

Database:

- Claims/Rules: PostgreSQL 15+

- Audit Logs: PostgreSQL with append-only tables

- Cache: Redis for session/external API responses

Why: ACID compliance for audit trail, proven at scale

External Data Integration:

API Gateway: httpx (async Python HTTP client)

Rate Limiting: Redis-based token bucket

Retry Logic: tenacity library

Batch Downloads:

- Scheduler: APScheduler (monthly NDC, quarterly CMS)

- Storage: S3-compatible object storage (MinIO/AWS S3)

ArqAI Governance Components

Audit Trail (ArqMesh™):

```
class AuditLog:
  """Immutable audit trail for all system actions"""

  def log_event(self, event_type, user, action, details):
    """
    Write to append-only PostgreSQL table
    Include cryptographic hash of previous entry for chain integrity
    """
    entry = {
      "timestamp": datetime.utcnow(),
      "event_id": uuid4(),
      "event_type": event_type,
      "user": user,
      "action": action,
      "details": json.dumps(details),
      "previous_hash": self.get_latest_hash(),
      "current_hash": self.calculate_hash(details)
    }

    # Write-only, no updates allowed
    db.execute(
      "INSERT INTO audit_log VALUES (...)",
      entry
```

```
    )
```

Rule Engine:

```
class RuleEngine:
  """Configurable fraud detection rules"""

  def __init__(self):
    self.rules = self.load_rules_from_db()

  def evaluate_claim(self, claim_data, external_data):
    """
    Apply all enabled rules to a claim

    Return: List of triggered rules with severity
    """
    triggered = []

    for rule in self.rules:
      if not rule['enabled']:
        continue

      # Dynamic rule execution
      result = self.execute_rule(
        rule_id=rule['id'],
        claim=claim_data,
        benchmarks=external_data,
        thresholds=rule['thresholds']
      )

      if result['triggered']:
        triggered.append({
          'rule_id': rule['id'],
          'severity': result['severity'],
          'evidence': result['evidence']
        })

    return triggered
```

Dashboard & UI

Frontend:

Framework: Streamlit (POC) → React + TypeScript (production)

Why: Streamlit = 5x faster POC development

Charts: Plotly for interactive visualizations

Tables: AG Grid for high-performance data tables

Deployment: Docker container, reverse proxy (nginx)

Dashboard Views:

```
# Streamlit POC structure
import streamlit as st

def main():
  st.set_page_config(layout="wide", page_title="ArqAI FWA Detection")

  # Sidebar navigation
  page = st.sidebar.selectbox(
    "Navigation",
    ["Executive Overview", "Case Investigation", "Compliance", "Admin"]
  )

  # Role-based access control
  user_role = authenticate_user()

  if page == "Executive Overview":
    show_executive_dashboard()
  elif page == "Case Investigation":
    show_investigation_workspace()
  elif page == "Compliance":
    show_compliance_dashboard()
  elif page == "Admin":
    if user_role == "admin":
      show_admin_configuration()
    else:
      st.error("Access denied: Admin privileges required")

def show_executive_dashboard():
  """High-level metrics and trends"""
  col1, col2, col3 = st.columns(3)

  with col1:
    st.metric("Total Fraud Identified", "$4.2M", "+$1.1M from last month")
  with col2:
    st.metric("Cases Under Investigation", "87", "-12 from last week")
  with col3:
    st.metric("Recovery Rate", "68%", "+5% from last quarter")

  # Interactive charts
  st.plotly_chart(create_fraud_trends_chart())
  st.plotly_chart(create_provider_risk_heatmap())
```

Data Sources Integration

CMS API Integration:

```
class CMSDataService:
    """Centralized CMS data access with caching"""

    def __init__(self):
        self.nppes_api = "https://npiregistry.cms.hhs.gov/api/"
        self.cache = redis.Redis()

    @retry(stop=stop_after_attempt(3), wait=wait_exponential())
    async def validate_npi(self, npi: str) -> dict:
        """
        Validate provider NPI against NPPES

        Cache results for 7 days
        """
        cache_key = f"npi:{npi}"
        cached = self.cache.get(cache_key)

        if cached:
            return json.loads(cached)

        async with httpx.AsyncClient() as client:
            response = await client.get(
                f"{self.nppes_api}?version=2.1&number={npi}"
            )

            if response.status_code == 200:
                data = response.json()
                self.cache.setex(cache_key, 604800, json.dumps(data))  # 7 days
                return data
            else:
                raise ValueError(f"NPI {npi} not found in NPPES")

    def load_ndc_directory(self):
        """
        Download and parse NDC directory (monthly)
        """
        url = "https://www.fda.gov/media/70203/download"
        df = pd.read_csv(url, dtype=str)

        # Transform to usable format
        ndc_lookup = df.set_index('NDC').to_dict('index')

        # Store in database
        self.db.bulk_upsert('ndc_reference', ndc_lookup)
```

Testing Strategy

Unit Tests (Per Component)

Rule Engine Tests:

```
def test_upcoding_detection():
    """Test M1: Upcoding rule"""

    # Setup test data
    claim = {
        'claim_id': 'TEST-001',
        'cpt_code': '99285',
        'diagnosis': 'E11.9',
        'amount_billed': 482.00
    }

    benchmark = {
        'cpt_code': '99285',
        'expected_cost': 145.00
    }

    # Execute rule
    result = rule_engine.execute_rule('M1', claim, benchmark)

    # Assert
    assert result['triggered'] == True
    assert result['severity'] > 2.0  # >200% overpayment
    assert 'upcoding' in result['evidence'].lower()
```

Audit Trail Tests:

```
def test_audit_trail_immutability():
    """Verify audit log cannot be modified"""

    # Write initial entry
    audit_log.log_event('test', 'user@test.com', 'action', {'key': 'value'})


    # Attempt to modify (should fail)
    with pytest.raises(PermissionError):
        db.execute("UPDATE audit_log SET user = 'hacker' WHERE event_id = ...")


    # Verify hash chain integrity

    entries = db.query("SELECT * FROM audit_log ORDER BY timestamp")
    for i in range(1, len(entries)):
        assert entries[i]['previous_hash'] == entries[i-1]['current_hash']
```

Integration Tests

End-to-End Detection Flow:

```
def test_full_fraud_detection_pipeline():
```

```
"""Test complete flow: claims → detection → investigation"""

    # 1. Ingest claims
    claims = load_test_claims('test_data/claims_sample.csv')
    ingestion_service.process_batch(claims)

    # 2. Enrich with external data
    enriched = enrichment_service.enrich(claims)

    # 3. Apply rules
    results = rule_engine.evaluate_all(enriched)

    # 4. Score risks
    scores = scoring_engine.calculate_scores(results)

    # 5. Generate evidence
    for case in scores:
        if case['risk_score'] > 60:
            evidence = explainability.generate_evidence(case)
            assert 'why_flagged' in evidence
            assert 'supporting_data' in evidence
            assert 'confidence_score' in evidence

    # 6. Verify audit trail
    audit_entries = audit_log.get_entries_for_batch(claims[0]['batch_id'])
    assert len(audit_entries) > 100  # Expect many logged events
```

Performance Tests

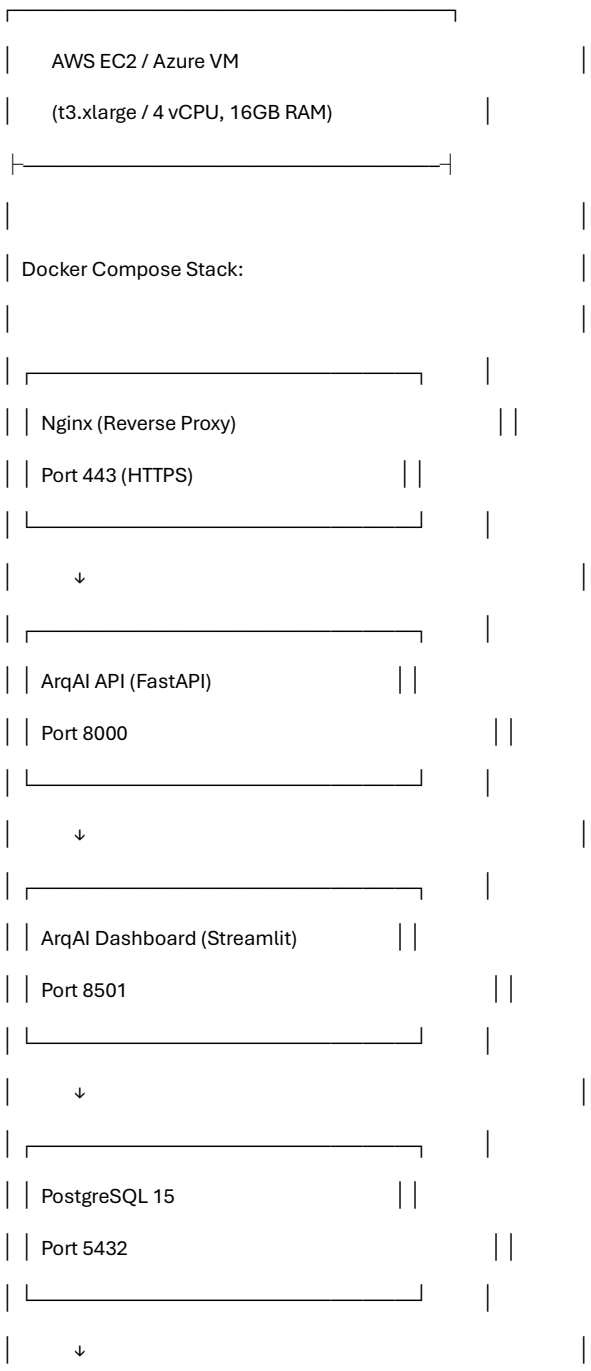
Batch Processing Benchmark:

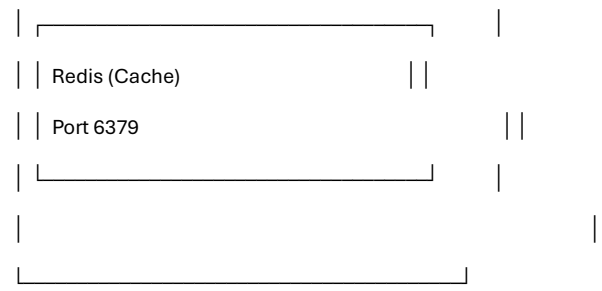
```
def test_batch_processing_performance():
    """Verify 10K claims processed in <30 minutes"""

    claims = generate_synthetic_claims(count=10000)

    start_time = time.time()
    results = fraud_detection_service.process_batch(claims)
    elapsed = time.time() - start_time

    assert elapsed < 1800  # 30 minutes
```

```
assert len(results) == 10000

print(f"Processed {len(claims)} claims in {elapsed:.2f}s")
print(f"Throughput: {len(claims)/elapsed:.1f} claims/sec")
```

Deployment Architecture

POC Deployment (Single Server)





Docker Compose Configuration:

```
version: '3.8'

services:
 nginx:
  image: nginx:alpine
  ports:
   - "443:443"
  volumes:
   - ./nginx.conf:/etc/nginx/nginx.conf
   - ./ssl:/etc/nginx/ssl
  depends_on:
   - api
   - dashboard

 api:
  build: ./backend
  environment:
   - DATABASE_URL=postgresql://user:pass@postgres:5432/arqai
   - REDIS_URL=redis://redis:6379
  depends_on:
   - postgres
   - redis

 dashboard:
  build: ./frontend
  environment:
   - API_URL=http://api:8000
  depends_on:
   - api

 postgres:
  image: postgres:15-alpine
  environment:
   - POSTGRES_DB=arqai
   - POSTGRES_USER=arqai_user
   - POSTGRES_PASSWORD=${DB_PASSWORD}
  volumes:
   - postgres_data:/var/lib/postgresql/data
```

```
  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data

volumes:
  postgres_data:
  redis_data:
```

Next Steps & Action Items

Immediate (Before Next Call with Sunil)

1. **Validate risk scoring approach**
    - Send proposed algorithm for feedback
    - Request example historical fraud cases for calibration
2. **Clarify data source access**
    - Confirm CMS API rate limits acceptable
    - Identify any proprietary data sources needed
3. **Finalize POC scope**
    - Confirm 10 rules sufficient for POC (vs all 29)
    - Agree on manual workflow complexity (simple vs full)

Week 1 Deliverables

4. **Technical architecture document** (THIS DOCUMENT)
    - Share with Sunil for final review
    - Incorporate feedback
5. **Data integration plan**
    - Document each external data source
    - API credentials and access procedures
6. **Development environment setup**
    - Infrastructure provisioning
    - Development tools and CI/CD

Week 2-4 Execution

7. **Follow implementation roadmap**
    - Weekly progress updates to Sunil
    - Demo intermediate milestones
8. **Prepare POC presentation materials**
    - Executive summary deck
    - Live demo script
    - ROI calculation methodology

Appendix: Rule Reference

Medical FWA Rules (16 Total)

Rule ID	Category	Fraud Type	Implementation Priority
M1	Upcoding	Fraud	**HIGH** (POC Phase 1)
M2	Unbundling	Fraud	MEDIUM (POC Phase 2)
M3	Duplicate Billing	Fraud	**HIGH** (POC Phase 1)
M4	Phantom Billing	Fraud	**HIGH** (POC Phase 1)
M5	Kickback/Self-Referral	Fraud	MEDIUM (POC Phase 2)
M6	Medically Unnecessary	Waste	MEDIUM (POC Phase 2)
M7	Provider Collusion	Fraud	LOW (Post-POC)
M8	Modifier Misuse	Fraud	MEDIUM (POC Phase 2)
M9	Copay Waiver	Abuse	LOW (Post-POC)
M10	Inpatient/Outpatient Misclass	Fraud	MEDIUM (POC Phase 2)
M11	DME Fraud	Fraud	MEDIUM (POC Phase 2)
M12	Lab/Diagnostic Abuse	Waste	MEDIUM (POC Phase 2)
M13	Provider Ghosting	Fraud	LOW (Post-POC)
M14	Double Dipping	Fraud	MEDIUM (POC Phase 2)
M15	Telehealth Fraud	Fraud	MEDIUM (POC Phase 2)
M16	Chart Padding	Abuse/Fraud	LOW (Post-POC)

Pharmacy FWA Rules (13 Total)

Rule ID	Category	Fraud Type	Implementation Priority
P1	Prescription Forgery	Fraud	MEDIUM (POC Phase 2)
P2	Doctor Shopping	Abuse	MEDIUM (POC Phase 2)
P3	Pharmacy Shopping	Abuse	LOW (Post-POC)
P4	Early Refill	Waste/Abuse	**HIGH** (POC Phase 1)
P5	Controlled Substance Diversion	Fraud/Abuse	MEDIUM (POC Phase 2)
P6	Phantom Claims	Fraud	**HIGH** (POC Phase 1)
P7	Upcoding/High-Cost Substitution	Fraud	MEDIUM (POC Phase 2)
P8	Kickback/Split Billing	Fraud	MEDIUM (POC Phase 2)
P9	Invalid Prescriber	Fraud	**HIGH** (POC Phase 1)
P10	Stockpiling/Hoarding	Waste	LOW (Post-POC)
P11	Compound Drug Fraud	Fraud	LOW (Post-POC)
P12	Phantom Members	Fraud	**HIGH** (POC Phase 1)
P13	Pharmacy-Provider Collusion	Fraud	LOW (Post-POC)

POC Phase 1 (Week 1-2): 7 rules implemented
POC Phase 2 (Week 3-4): Additional 5-8 rules added
Post-POC: Remaining rules for production deployment

Document Control

Version: 2.0 (Post-Sunil Review)
Last Updated: October 31, 2025
Next Review: November 7, 2025 (Post-implementation kickoff)
Owner: Habib Mehmoodi, ArqAI
Reviewer: Sunil Pal (Advisor - Healthcare Domain Expert)

Change Log:

- v1.0: Initial architecture (pre-Sunil review)
- v2.0: Major updates based on 10/31 technical review:
    - Added configurable rule engine with admin UI

- Removed HIPAA/anti-discrimination as separate checks
- Added post-investigation workflow engine
- Documented 29 specific rules (medical + pharmacy)
- Added external data source integration details
- Defined risk scoring algorithm
- Clarified continuous processing model

Outstanding Questions for Sunil:

1. Risk scoring methodology validation
2. Rule weight calibration
3. Data source update frequency
4. Manual workflow complexity for POC
5. Admin UI scope (modify vs add rules)
6. Claim volume assumptions
