Pythia Platform - Proactive Insights Engine AI Architecture

Architecture Overview

High-Level Architecture Components

Data Sources Layer

- CRM/ERP
- Marketing Data
- Event Streams
- Unstructured

Ingestion & Queue Layer

- Kafka / RabbitMQ / BullMQ
- Multi-tenant event routing
- Rate limiting
- Dead letter queues

Data Processing Pipeline

- **Data Validation**: Schema checks
- **Feature Engineering**: Transformations
- **Embeddings Gen**: Vector creation
- **Feature Store**: Redis/Feast

AI Model Services

- **LLM Service**: GPT-4/Claude, Summarization
- **Anomaly Detection**: Time-series models, Statistical analysis
- **Predictive Scoring**: Churn/Lead models, XGBoost/Neural nets

Storage & Cache Layer

- PostgreSQL
- Vector DB
- Redis Cache
- S3/Blob

API Layer

- Node.js API Gateway + FastAPI Services

Key Design Principles

- Multi-tenant isolation at data and compute layers
- Asynchronous processing for scalability
- Model-agnostic architecture for flexibility
- Comprehensive observability and monitoring

Technology Stack

- FastAPI for ML services, Node.js for API gateway
- PostgreSQL + pgvector for embeddings
- Redis for caching and feature store
- Kafka/BullMQ for event streaming

Data Pipeline Design

Multi-Tenant Event Ingestion

- **Architecture**: Kafka topics partitioned by tenant_id with dedicated consumer groups
- **Rate Limiting**: Token bucket algorithm per tenant (configurable limits)
- **Schema Validation**: Pydantic models with versioned schemas per tenant
- **Dead Letter Queue**: Failed events routed to DLQ for manual review

Data Preprocessing Pipeline

Structured Data (PostgreSQL)

- SQL aggregations for time-series features
- Window functions for rolling metrics
- Indexed queries with query optimization
- Materialized views for common patterns

Unstructured Data (JSON/Text)

- Text cleaning and normalization
- Entity extraction (NER)
- Embedding generation (OpenAI/SentenceTransformers)
- Chunking for large documents

Feature Engineering Strategy

Feature Store: Redis + Feast

Online features served from Redis with TTL-based expiration. Offline features in PostgreSQL for model training.

Feature Types:

- **Real-time**: Last 24h metrics, rolling averages
- **Batch**: Historical aggregates, seasonal patterns
- **Derived**: Ratios, deltas, z-scores
- **Embeddings**: Vector representations cached for 7 days

Embedding Strategy

Generation

OpenAI text-embedding-3-large for semantic search. Batch processing for cost efficiency.

Storage

PostgreSQL pgvector with HNSW indexes. Partitioned by tenant_id.

Retrieval

Approximate nearest neighbor search. Filtered by tenant + metadata.

Model Selection & Strategy

Model Type Selection Framework

Use Case	Model Type	Reasoning	Latency
Text Summarization	LLM (GPT-4)	Complex context understanding needed	2-5s
Simple Classification	Rule Engine	Deterministic, fast, explainable	<50ms
Anomaly Detection	Statistical ML	Pattern-based, cost-effective	100-300ms
Churn Prediction	XGBoost/Neural	Structured features, high accuracy	50-200ms
Semantic Search	Embeddings + Vector DB	Similarity-based retrieval	50-150ms

LLM Services

- **Primary**: Azure OpenAI GPT-4
- **Backup**: Anthropic Claude
- **Use Cases**:
  - Executive summaries
  - Natural language insights
  - Anomaly explanations
  - Recommendation generation
- **Cost**: $0.01-0.03 per insight

Statistical/ML Models

- **Algorithms**:
  - Isolation Forest (anomalies)
  - ARIMA/Prophet (forecasting)
  - Z-score analysis
  - Rolling statistics
- **Training**: Weekly retraining
- **Cost**: ~$0.001 per prediction

Predictive Models

- **Frameworks**:
  - XGBoost (tabular data)
  - LightGBM (large scale)
  - PyTorch (deep learning)
  - Scikit-learn (baselines)
- **Training**: Daily for critical models
- **Cost**: ~$0.002 per prediction

Trade-off Analysis

Latency vs Accuracy

- Real-time alerts: Use cached features + fast models (<500ms)
- Daily digests: Allow complex LLM processing (5-10s acceptable)
- Implement tiered SLAs based on insight priority

Cost vs Quality

- LLMs for high-value insights (C-level summaries)
- Rule engines for simple classifications
- Batch processing to reduce API costs by 40%

Model Serving & MLOps

Hosting Architecture

FastAPI Microservices

- Separate service per model type
- Docker containers on Kubernetes
- Horizontal pod autoscaling (HPA)
- Health checks and readiness probes
- gRPC for internal service communication

Serverless Options

- Azure Functions for infrequent models
- Cold start mitigation with warm pools
- Lambda for burst traffic handling
- Cost-effective for low-traffic tenants

Model Versioning Strategy

Semantic Versioning: MAJOR.MINOR.PATCH (e.g., v2.1.3)

- **Model Artifacts**: Stored in Azure Blob Storage / S3 with version tags
- **Metadata**: Training metrics, feature schemas, dependencies in PostgreSQL
- **Registry**: MLflow Model Registry for tracking and lineage

Deployment Strategy

Canary Deployment (Preferred)

- Deploy new version to 5% of traffic
- Monitor error rates, latency, business metrics for 2 hours
- Gradually increase to 25% → 50% → 100% over 24 hours
- Automated rollback if error rate > 2% or latency p95 > 2x baseline

Shadow Deployment (Testing)

- Run new model alongside production without serving results
- Compare predictions for accuracy and consistency
- Validate performance under production load
- No user impact during validation period

Rollback Strategy

- Keep last 3 versions in hot standby
- Instant traffic switching via feature flags
- Automated alerts trigger rollback workflows
- Manual override available via ops dashboard

CI/CD Pipeline

GitHub Actions Workflow:

1. Code push triggers automated tests (unit + integration)
2. Model evaluation on holdout dataset (accuracy, latency, bias metrics)
3. Performance regression checks vs. production baseline
4. Docker image build and push to registry
5. Deploy to staging environment
6. Smoke tests and API contract validation
7. Manual approval gate for production
8. Canary deployment with automated monitoring

Quality Gates:

- Test coverage > 80%
- Model accuracy within 5% of baseline
- Inference latency p95 < 500ms for critical paths
- No high/critical security vulnerabilities

Monitoring & Observability

Model Performance Metrics

- Prediction accuracy (per tenant, per model)
- Feature drift detection (KL divergence)
- Concept drift via performance degradation
- Prediction confidence distributions
- False positive/negative rates

Infrastructure Metrics

- Request rate, latency (p50, p95, p99)
- Error rates by service and endpoint
- Resource utilization (CPU, memory, GPU)
- Queue depths and processing lag
- Cost per prediction by model type

Tech Stack: Prometheus + Grafana for metrics, OpenTelemetry for distributed tracing, Sentry for error tracking, custom dashboards for model-specific KPIs

Node.js API Integration

Communication Pattern: REST API with async/await, gRPC for high-throughput internal calls

```
// Example integration
const response = await fetch('http://ml-service:8000/predict', {
  method: 'POST',
  headers: { 'X-Tenant-ID': tenantId },
  body: JSON.stringify(features)
});
```

- Circuit breaker pattern for resilience (using Opossum)
- Request timeout: 10s for LLMs, 2s for ML models
- Retry logic with exponential backoff
- Response caching in Redis (5-60 min TTL)

AI Governance & Safety

Prompt & Template Management

Versioning Strategy

- Prompts stored in PostgreSQL with version tracking
- Git-based prompt engineering workflow
- A/B testing framework for prompt optimization
- Rollback capability for problematic prompts

Template Structure

```
System: You are an AI analyst for {tenant_name}...
Context: {dynamic_data}
User: Generate insights about {topic}
```

- Parameterized templates with validation
- Token counting to prevent context overflow
- Escape handling for special characters

Safety Filters & Content Moderation

Input Validation

- SQL injection prevention
- Prompt injection detection
- PII redaction before LLM calls
- Input length limits
- Malicious content filtering

Output Validation

- Fact-checking for critical insights
- Toxicity detection (Azure Content Safety)
- Hallucination detection via confidence scores
- Structured output validation
- Bias detection in predictions

Moderation Pipeline: Pre-LLM filter → LLM generation → Post-processing validation → Human review for flagged content (async)

Tenant-Level Isolation

Data Layer

- Row-level security (RLS) in PostgreSQL
- Tenant_id in all queries
- Separate schemas per tenant for high-value clients

Compute Layer

- Request context carries tenant_id
- Separate queues per tenant tier
- Resource quotas and rate limits

Model Layer

- Tenant-specific fine-tuned models (optional)
- Feature namespace isolation
- Prediction logs partitioned by tenant

Critical: No Cross-Tenant Data Leakage All API calls include X-Tenant-ID header. Middleware validates and injects tenant context. Audit logs track all data access.

Data Privacy & Compliance

Privacy Controls

- GDPR compliance: Data minimization, right to deletion
- PII detection and masking before model inference
- Encryption at rest (AES-256) and in transit (TLS 1.3)
- Data retention policies (90 days for logs, configurable for features)
- Consent management integration

Audit & Compliance

- Full audit trail for all predictions
- Model lineage tracking (data → features → predictions)
- Explainability reports (SHAP values for ML models)
- Regular bias audits across demographic segments
- SOC 2 Type II compliance documentation

LLM Provider Considerations:

- Azure OpenAI preferred (data not used for training)
- Business Associate Agreement (BAA) for healthcare tenants
- Data residency controls (EU data stays in EU regions)
- Option to use self-hosted models for sensitive clients

Cost Management & Optimization

Cost Tracking

- Per-tenant cost attribution
- LLM token usage monitoring
- Compute cost allocation by model type
- Daily budget alerts

Optimization Strategies

- Aggressive caching (60%+ cache hit rate)
- Batch processing for non-urgent insights
- Model tiering (GPT-3.5 for simple tasks)
- Prompt compression techniques

Scaling Strategy

Horizontal Scaling

- Kubernetes HPA based on queue depth
- Replica sets for high-traffic models
- Multi-region deployment for low latency

Performance Optimization

- Model quantization (8-bit inference)
- ONNX runtime for faster inference
- GPU instances for neural networks

Load Management

- Priority queues (premium vs standard)
- Request throttling per tenant tier
- Graceful degradation to simpler models