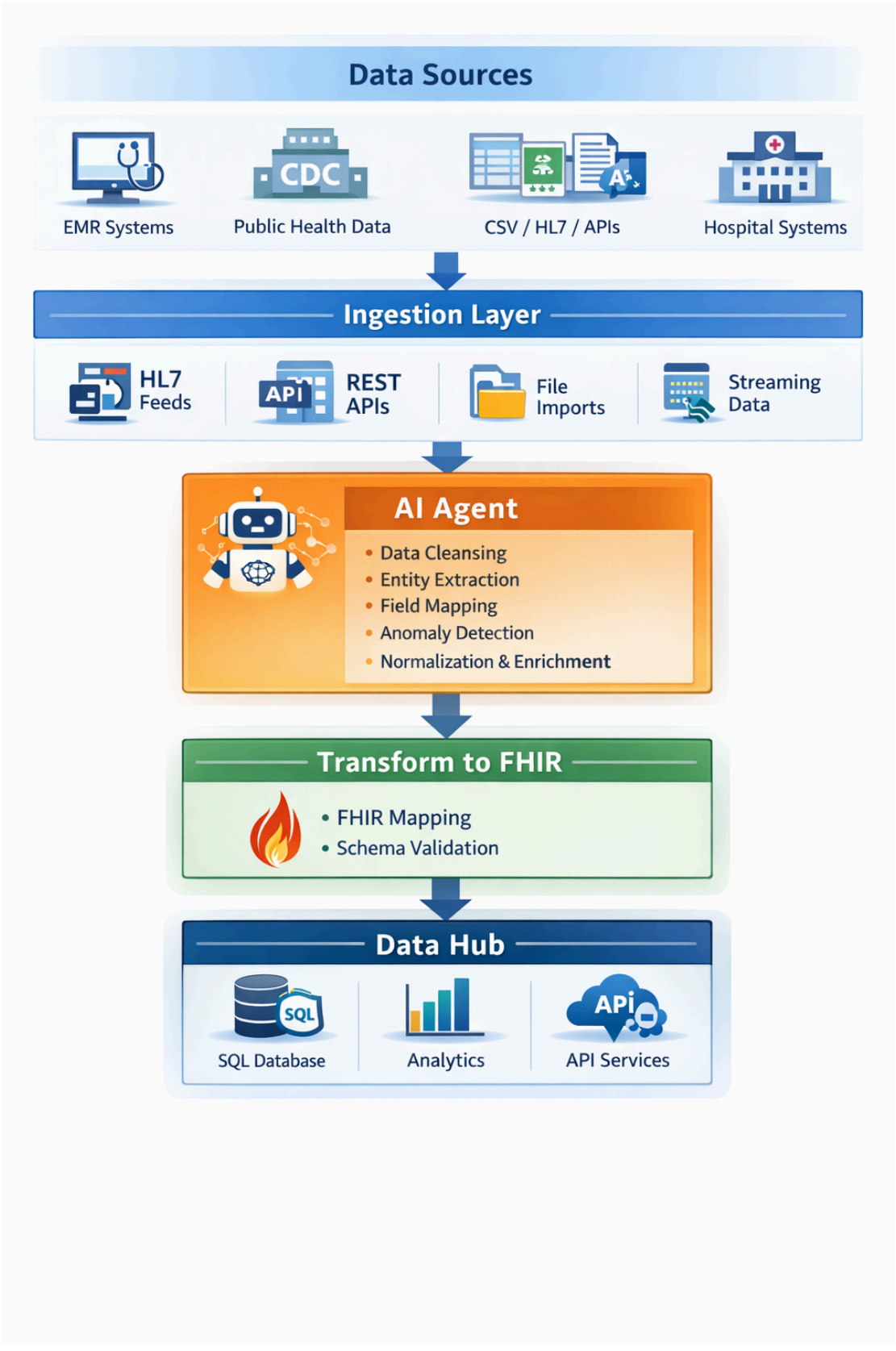
PyxGen
PORTFOLIO

Lowell Buschert

Interoperability Engineer

1-321-352-9679 lbuschert@pyxgen.com

Pipeline Component Review



PROBLEM

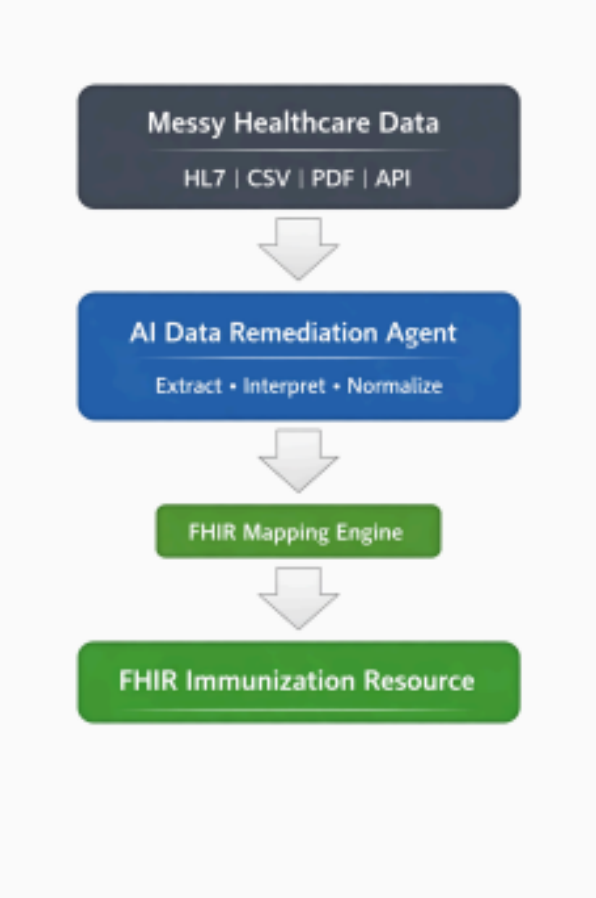
Healthcare data is rarely clean or consistent. It is typically scattered across structured, semi-structured, and unstructured sources such as HL7 messages, CSV extracts, PDFs, API feeds, and database exports. Within clinics, hospitals, and public health agencies, this data often lacks consistent structure or context, making it difficult to analyze, share, or use for meaningful decision-making.

Yet this data holds tremendous value — if it can be organized, standardized, and interpreted correctly. Traditional ETL pipelines can help move data between systems, but they often struggle with the realities of healthcare data: inconsistent formatting, incomplete fields, and unstructured documents that require interpretation rather than simple transformation.

To address this challenge, I developed a prototype **Python-based data remediation and transformation framework** that incorporates **AI-assisted agents** alongside traditional ETL processes. These agents help interpret messy data sources, extract key information from unstructured inputs such as PDFs or reports, and apply normalization rules before mapping the results to **FHIR-compliant resources**.

The result is a structured, analyzable dataset that can support clinical analytics, public health reporting, and interoperability initiatives — turning fragmented healthcare data into information that systems and clinicians can actually use.

Architecture Review



SOLUTION

AI-Assisted Healthcare Data Remediation and Interoperability Framework

This prototype implements a Python-based healthcare data remediation and transformation framework that combines traditional ETL processes with AI-driven data remediation agents. The system ingests diverse healthcare data sources—including HL7v2 messages, flat files (CSV, JSON, XML), PDFs, and API feeds—and transforms them into FHIR-compliant resources suitable for analytics, interoperability, and reporting.

Unlike traditional ETL pipelines that rely solely on deterministic transformation rules, this framework introduces an AI remediation layer that helps interpret messy or inconsistent healthcare data before mapping it to standards-based structures.

Originally designed for immunization data, the architecture is extensible and can support additional clinical domains such as admissions, encounters, procedures, lab results, and population health indicators. This makes the framework adaptable for hospitals, clinics, dental networks, public health systems, and research organizations.

Key Architectural Features

AI Data Remediation Layer

AI agents analyze raw healthcare data sources and assist in extracting, interpreting, and normalizing inconsistent or semi-structured inputs such as PDFs, reports, or irregular HL7 segments. This helps resolve common issues such as missing context, inconsistent formatting, and ambiguous field mappings.

Modular Processing Architecture

The framework separates data ingestion, AI remediation, transformation, and loading into modular components. Each stage can be tested independently and extended to support new data sources or FHIR resource types.

Standards-Driven Interoperability

The pipeline adheres to HL7v2 and FHIR standards, ensuring compatibility with both legacy health IT systems and modern interoperability platforms.

Centralized Data Dictionary and Mapping Logic

A schema-driven mapping approach ensures consistent and maintainable transformation logic across clinical domains while enabling transparent governance of field mappings.

Analytics-Ready Output

The final output consists of structured FHIR resources (Immunization, Patient, Encounter, Observation, etc.) that can be consumed by analytics platforms, dashboards, public health

reporting systems, or machine learning models.

Directory Structure Overview

```
healthcare_ai_pipeline/
 ├── README.md ← Project summary, architecture overview, installation instructions ├──
 data/ ← Sample HL7v2, CSV, JSON, and document-based test files ├── agents/
 │  └── remediation_agent.py ← AI-assisted extraction and normalization layer
 ├── pipeline/
 │  ├── ingest.py ← Reads input from files, folders, or API endpoints │
 ├── transform.py ← Maps normalized data to FHIR resources
 │  └── load.py ← Writes output to storage, databases, or FHIR servers ├──
 tests/
 │  └── test_transform.py ← Unit tests for transformation logic
 └── notebooks/
    └── demo_pipeline.ipynb ← Jupyter walkthrough for technical and
non-technical stakeholders
```

SUMMARY

This project demonstrates how lightweight Python tooling combined with AI-assisted data remediation can create practical healthcare interoperability infrastructure in fragmented environments.

The framework ingests messy clinical data from multiple formats and sources, applies remediation and normalization logic, and outputs FHIR-compliant resources ready for analytics, reporting, and interoperability workflows.

Because the architecture is modular and standards-driven, it supports rapid prototyping as well as operational use in clinical, public health, and research settings. The design also allows AI agents to assist with interpreting unstructured or inconsistent data before it enters the standards-based transformation layer.

VALUE

This framework enables healthcare organizations to unlock the value of fragmented clinical data by transforming and standardizing disparate sources into interoperable datasets.

Population Health & Risk Stratification

Aggregates and normalizes multi-source clinical data to support identification of at-risk cohorts, chronic disease management, and population-level care coordination.

Preventive Care & Gaps in Care

Analyzes structured clinical histories to detect incomplete care interventions such as missed screenings, vaccinations, or follow-up visits—supporting proactive outreach and compliance programs.

Programmatic & Regulatory Reporting

Produces jurisdiction-ready datasets aligned with CDC, PHAC, and provincial/state reporting requirements for immunizations, communicable disease tracking, and chronic condition monitoring.

Operational & Clinical Dashboards

Feeds normalized clinical data into analytics platforms and visualization tools to support clinical KPIs, operational metrics, and patient engagement insights.

Predictive Analytics & Machine Learning

Provides clean, structured longitudinal datasets suitable for advanced analytics and machine learning workflows, including care quality modeling, demand forecasting, and clinical trend analysis.

TECH STACK

The system leverages lightweight, proven Python tooling designed for portability, transparency, and rapid deployment in constrained environments.

Core Language

Python 3.10+

Data Handling

pandas, datetime, csv, json for parsing, transformation, and structured data manipulation.

Healthcare Standards Libraries

hl7apy, hl7 for HL7 v2 parsing and message handling
fhir.resources or custom mapping modules for FHIR R4 resource generation

AI-Assisted Data Processing

Python-based agents for extraction, interpretation, and normalization of semi-structured or unstructured healthcare inputs such as documents, reports, or irregular message structures.

Web / API Integration

Flask or FastAPI for optional ingestion endpoints, service interfaces, or FHIR proxy services.

Storage Options

Local
SQLite, CSV, JSON for lightweight deployments

Enterprise
PostgreSQL, MySQL, or FHIR servers such as HAPI FHIR or Microsoft FHIR Server.

Visualization & Testing
Jupyter Notebooks for demonstration, debugging, and stakeholder walkthroughs
pytest for unit testing of transformation and mapping logic
Atlassian Loom for technical walkthroughs and demonstrations

The technology stack is intentionally open source, portable, and lightweight, minimizing licensing overhead while enabling rapid experimentation and deployment.

STANDARDS & REFERENCE MODELS

The framework leverages established healthcare interoperability standards while allowing customizable mappings to support diverse environments.

HL7 v2.x

Supports ingestion of inbound messages such as ADT, ORU, and VXU from EMRs, registries, and HIE systems with detailed segment and OBX parsing.

FHIR R4 / US Core / Canadian Baseline

Exports structured FHIR resources including Immunization, Observation, Condition, Patient, and Encounter for interoperability platforms and analytics environments.

Clinical Terminology Systems

Optional normalization and mapping using established clinical vocabularies including:

SNOMED CT
LOINC
CVX / MVX vaccine codes
CDC and PHAC reporting vocabularies

Jurisdictional and Custom Mapping
The architecture supports configurable mapping logic to accommodate province/state-specific reporting requirements, clinical pathways, and organizational standards without hardcoding transformations.

EXAMPLES

This workflow reflects a modern healthcare data pipeline

```
|Data Sources (EMR / CDC / CSV / API)
▼
|Ingestion Layer (fetch_raw_data)
▼
|AI Agent Processing (data normalization / inference)
▼
|FHIR Transformation
▼
Data Persistence (MySQL / Analytics)
```

```
# Example Main ETL AI Agent Controller Loop
def run_agent(input_data):
 max_steps = 20
 steps = 0
 state = {
     "file_path": input_data,
     "raw_data": None,
```

```
        "fhir_patient": None,
        "hl7": None,
        "patient": None,
        "fhir_resources": [],
        "persisted": False,
        "complete": False,
        "history": []    }
    tool_name = "start"
    while tool_name != "end":
        print("\nRUNNING:", tool_name)
        tool_name = ask_agent(state)
        print(steps,"  ",tool_name)
        tool = TOOLS[tool_name]["function"]
        tool_name = tool(state)
        state["history"].append(tool_name)
        steps += 1
    test_end = state.get("end")
    test_report = state.get("selected_report")
    print("\nFINAL STATE BEFORE REPORT:")
    for k, v in state.items():
        print(k, "=", type(v))
    print(test_end,"  ",test_report)
    if not state.get("end") and state.get("selected_report"):
       state = generate_report(state)
    print("\nPIPELINE COMPLETE\n")
if __name__ == "__main__":


  #  run_agent("if __name__ == "__main__":
```

```
print("\nAI ETL Pipeline\n")

file_path = input("Enter raw data file (HL7): ")

run_agent(file_path)
```

Sample Raw Data:

1007649,PELOT PHARMACY,831 MANATEE AVE
E,,BRADENTON,FL,342081243,,,27.495385,-82.554966,POINT (-82.554966
27.495385),2024-01-25
00:00:00+00:00,true,false,,false,false,,false,false,false,false,false,false,false,false,false,false,fals e,,

Sample Human Readable Output + JSON