IntelliHealth: Your AI-Driven Guide to Smarter Health Plan Choices

By
Anoushka Bansal
Nishita Vikas Shewale

Index	
Sr No.	Content
1	Last Updated
2	Project Overview
3	Summary
4	Architecture and Tech Stack
5	Core Features - Patient Profile Management - Insurance Plan Retrieval - Rule-Based Filtering (Neo4j) - LLM-Based Dynamic Scoring - Plan Distribution & Metrics
6	Installation Instructions
7	API Endpoints Overview
8	Development Log (April 2025)
9	Why IntelliHealth Stands Out
10	Future Enhancements
11	Contact

1. Last Updated: 25th April 2025

2. Project Overview

IntelliHealth is an AI-driven backend platform that intelligently recommends U.S. health insurance plans based on a patient's demographics, healthcare needs, and financial situation. It integrates multiple databases (PostgreSQL, Snowflake, Neo4j) and Large Language Models (LLMs) to deliver highly personalized and explainable recommendations.

3. Summary

The Problem:

- Health insurance plans are confusing, leading to **overpaying**, **under-insurance**, or **no coverage**.
- Surveys show **45% of Americans** find plan documents confusing, and **60%** have issues using insurance.

The IntelliHealth Solution:

A **FastAPI-powered AI platform** that:

- Collects **patient demographics** (age, health, budget, family needs) — stored in **PostgreSQL**.
- Fetches **insurance plans** from **Snowflake** — using dynamic SQL filtering based on patient needs.
- Applies **rule-based reasoning** (e.g., Diabetes rules, Maternity rules) in **Neo4j** graphs to filter plans.
- Uses **LLMs (GPT-4o, Claude)** to **score and rank** plans via **Chain-of-Thought (CoT)** reasoning.
- Provides a **Streamlit UI** for easy, clean interaction.
- Fully **Dockerized** for scalable deployments.

How It Works:

1. **Patient Profile Creation** → FastAPI + PostgreSQL
2. **Insurance Plan Retrieval** → Snowflake + Dynamic SQL Filtering
3. **Enhanced Filtering** → Neo4j Rule-Based Reasoning
4. **Dynamic Scoring** → LLMs using CoT to rank plans
5. **UI and API Integration** → Streamlit frontend + FastAPI backend

4. Architecture and Tech Stack

Layer	Technology
Backend	FastAPI (Python)
Databases	PostgreSQL, Snowflake, Neo4j
LLMs	OpenAI GPT-4o, GPT-o3, Claude (via Snowflake Cortex)
DevOps	Docker Compose, GitHub Actions (CI/CD)
Frontend	Streamlit
Libraries	SQLAlchemy, Pydantic, Pandas, Neo4j Driver, Snowflake Connector

5. Core Features

Patient Profile Management

- Store and manage patient profiles using PostgreSQL.
- CRUD APIs with FastAPI and SQLAlchemy.
- Data validation using Pydantic models.

Insurance Plan Retrieval

- Fetch, normalize, and filter plans from Snowflake.
- Tailored SQL queries based on patient attributes (state, budget, family status, conditions).
- Normalize results to clean JSON format for API responses.

Rule-Based Filtering (Neo4j)

- Apply graph-based reasoning rules (e.g., Diabetes, Maternity, Family Coverage).
- Filter plans using dynamic median thresholds by PlanType.
- Store patient-plan relationships in Neo4j for symbolic reasoning.

LLM-Based Dynamic Scoring

- Use Chain-of-Thought (CoT) prompting to rank filtered plans.
- LLMs evaluate plan relevance based on patient profiles.
- Scoring is modular, explainable, and model-agnostic.

Plan Distribution & Metrics

- Summarize how many plans satisfy how many rules.
- Provide PlanType distribution to assist patient choices (HMO, PPO, etc.)

6. Installation Instructions

1. **Clone Repository:**

   ```
   git clone https://github.com/<your-repo-org>/intellihealth.git

   cd intellihealth
   ```

2. **Set Up Environment Variables:**

   ```
   cp .env.example .env

   # Populate .env with your PostgreSQL, Snowflake, and Neo4jcredentials
   ```

   Example .env snippet:

   ```
   DATABASE_URL=postgresql://user:password@localhost:5432/intellihealth_
   db
   SNOWFLAKE_USER=your_user
   SNOWFLAKE_PASSWORD=your_password
   NEO4J_URI=bolt://neo4j:7687
   NEO4J_AUTH=neo4j/neo4jpassword
   ```

3. **Run with Docker Compose:**

   ```
   docker-compose up --build
   ```

4. **Access the Streamlit UI:**

   ```
   http://localhost:8501
   ```

7. API Endpoints Overview

Endpoint	Method	Purpose
`/patients/`	POST	Create new patient
`/insurance-plans/`	GET	Retrieve all plans
`/filter-plans/`	POST	Filter plans via Snowflake
`/process-plans/`	POST	Apply Neo4j rules to plans
`/plan-distribution/`	GET	PlanType distribution overview
`/recommend-insurance/`	POST	LLM-based final scoring

8. Work Log Summary (As of April 2025)

Task	Subtask	Assigned To	Status
Project Setup	Initialize Git repository	Nishita-Shewale	Completed
	Set up FastAPI structure	Nishita-Shewale	Completed
	Configure Docker Compose	AnoushkaBansal	Completed
Database Setup - PostgreSQL	Define Patient model	AnoushkaBansal	Completed
	Add Alembic migrations	AnoushkaBansal	Completed
Database Setup - Snowflake	Implement connection and normalization		Completed
	Populate with cleaned_plans_data.csv	Nishita-Shewale	Completed
	Validate data integrity	Nishita-Shewale	Completed
	Align schema with InsurancePlan	AnoushkaBansal	Completed
Database Setup - Neo4j	Set up driver	AnoushkaBansal	Completed
	Document schema	Nishita-Shewale	Completed
Data Processing	Clean CSV data	AnoushkaBansal	Completed
	Upload to Snowflake	Nishita-Shewale	Completed
	Enhance normalization	AnoushkaBansal	Completed
API Development - Patient Endpoints	PUT /patients/{patient_id}	AnoushkaBansal	Completed
	DELETE /patients/{patient_id}	AnoushkaBansal	Completed
	Enhance POST /patients/	Nishita-Shewale	Completed
API Development - Insurance Plan Endpoints	Ensure pagination consistency	AnoushkaBansal	Completed
Error Handling	Add logging	Nishita-Shewale	Completed
Rule Engine	Expand rule set	AnoushkaBansal, Nishita-Shewale	Completed
	Optimize Neo4j queries	AnoushkaBansal	Completed
	Integrate CoT reasoning	Nishita-Shewale	Completed
Testing	Unit Tests	AnoushkaBansal, Nishita-Shewale	Completed
	Integration Tests	Nishita-Shewale	Completed

	LLM Validation	AnoushkaBansal	Completed
Documentation	Draft README	Nishita-Shewale	Completed
	Enhance Setup Instructions	Nishita-Shewale	Completed
	API Examples	AnoushkaBansal	Completed
	Research Paper Support	Nishita-Shewale	Completed
UI Development	Streamlit UI	Nishita-Shewale, AnoushkaBansal	Completed
Deployment	Production Server	Nishita-Shewale	Completed
	CI/CD	AnoushkaBansal	Completed
Enhancements	Authentication	AnoushkaBansal	Completed
	Scalability	Nishita-Shewale	Completed

9. Challenges Faced

- **Data Volume and Complexity:**
  - Ensuring schema alignment between Snowflake and PostgreSQL models dynamically
- **LLM Integration:**
  - Designing effective Chain-of-Thought prompts that are consistent across different models (GPT-4o, o3, Claude)
  - Handling non-deterministic outputs in scoring which made debugging difficult
- **Graph Database Reasoning:**
  - Modeling complex, multi-condition eligibility rules in Neo4j while maintaining query efficiency
- **System Scalability:**
  - Building Dockerized environments that could smoothly scale and support multiple concurrent users.
- **End-to-End Testing:**
  - Designing integration tests that simulate real-world patient profiles and plan recommendations.
  - Handling edge cases like missing fields, null values, and rollback scenarios during API testing.
- **Frontend-Backend Sync:**
  - Ensuring Streamlit UI correctly handles backend validation errors and displays user-friendly error messages.

10. Why IntelliHealth Stands Out

- **Transparent:** Users can see how decisions are made (graph links + LLM reasoning)
- **Personalized:** Combines static rules + flexible LLMs for deep personalization.
- **Modular:** Easy to extend with new rules, new models, or data sources.
- **Scalable:** Streamlined for production with Docker, Redis caching, and GitHub Actions CI/CD

11. Future Enhancements

- Auto-plan enrolment triggers.
- Smart alerts for plan changes based on lifestyle updates.
- Expanding support for multilingual patient profiles.

12. Contact

- **Anoushka Bansal:** bansal.ano@northeastern.edu
- **Nishita Vikas Shewale:** shewale.n@northeastern.edu

GitHub: https://github.com/Nishita-Shewale/Intellihealth-Health-Plan-Advisor