IRIS EHR Agent

## Abstract

Hospitals need to convert patient notes into standard billing codes (ICD-9 for diagnoses, CPT for procedures). Currently, human coders read medical notes and manually find matching codes - this is slow, expensive, and blocks hospital revenue.

The IRIS Medical Coding Assistant uses AI to automate this task: it searches through previously-coded notes to find similar cases and suggests codes instantly. When it's not confident, it uses a smart AI model to reason through the problem. This creates fast, reliable code assignment that helps hospitals get paid faster and reduces errors.

# Background -

TechCare Solutions is a healthcare technology provider working with multiple hospital clients. Currently, these hospitals:

* **Manually map** patient notes and look up medical codes in books/databases
* **Spend hours per record**, creating huge delays
* **Experience inconsistent coding**, that lead to billing problems and compliance issues
* **Can't find enough trained medical coders,** making operations expensive

To stay competitive, TechCare's clients need an automated system that can quickly read patient notes, find similar past examples, and suggest accurate codes - freeing their staff to focus on patient care.

## Objective

* Build an AI-powered assistant that:
  + Automatically reads patient notes and suggests the correct ICD-9 and CPT codes
  + Uses Google Cloud AI tools for fast search through historical records
  + Helps hospitals code patients faster and more accurately

# Problem Statement

TechCare’s HLS clients struggle with:

1. **Manual bottlenecks** in code assignment that delay reimbursements
2. **High error rates** from inconsistent coding interpretations
3. **Scalability challenges** during patient encounter surges

**Your task:** Build a smart coding assistant using Google Cloud that:

* Takes patient notes and finds similar past cases
* Uses Google Cloud AI to understand and search through medical records
* Suggests codes based on what worked for similar patients
* Uses advanced AI reasoning when it can't find good matches
* Provides a simple web interface for hospital staff
* Tracks how well it's working with dashboards and alerts

# Data

|  |  |  |
| --- | --- | --- |
| Dataset | Records | Purpose |
| source\_records.csv | 1 ,976 notes with JSON-encoded ICD-9 & CPT mappings | Ground-truth corpus for retrieval and evaluation |
| Vector index (pre-loaded) | 15 K note embeddings | Enables instantaneous top-K similarity search |
| Loader scripts | Python notebooks | Push vectors and metadata to Vertex AI Matching Engine |

**1. Source Records Dataset (source\_records.csv)**

This dataset contains 1,976 clinical records with 15 fields, including clinical documentation (radiology reports and medical examination notes) with comprehensive medical code mappings. The dataset includes patient identifiers, timestamps, clinical categories, detailed clinical text, and JSON-formatted arrays of ICD-9 diagnoses, procedures, and CPT codes with complete descriptions.

**Data Dictionary - Source Records:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Description** | **Sample Value** |
| row\_id | Integer | Unique identifier for each clinical record | 1, 2, 3... |
| SUBJECT\_ID | Integer | Anonymous patient identifier | 3, 5, 7... |
| HADM\_ID | Integer | Hospital admission identifier | 145834, 142345... |
| CHARTDATE | String | Date when clinical documentation was created | "2101-10-26" |
| CHARTTIME | String | Date and time of documentation | "2101-10-26 6:01" |
| STORETIME | String/Null | Storage timestamp (mostly null) | null |
| CATEGORY | String | Clinical department or service category | "Radiology", "Nursing", "Physician" |
| DESCRIPTION | String | Brief description of the clinical procedure/exam | "CHEST (PORTABLE AP)", "CARDIAC CATH" |
| CGID | Float/Null | Caregiver identifier (mostly null) | null, 1234.0 |
| ISERROR | String/Null | Error flag for documentation (mostly null) | null, "Y" |
| TEXT | String | Complete clinical documentation including radiology reports, examination notes, medical findings, and impressions | "[**2101-10-26**] 6:01 AM CHEST (PORTABLE AP)... IMPRESSION: These findings are consistent with moderate left heart failure..." |
| diagnoses | String (JSON) | Array of ICD-9 diagnosis codes with descriptions | "[{'ICD9\_CODE': '2639', 'SHORT\_TITLE': 'Protein-cal malnutr NOS', 'LONG\_TITLE': 'Unspecified protein-calorie malnutrition'}...]" |
| procedures | String (JSON) | Array of ICD-9 procedure codes with descriptions | "[{'ICD9\_CODE': 9604, 'SHORT\_TITLE': 'Insert endotracheal tube', 'LONG\_TITLE': 'Insertion of endotracheal tube'}...]" |
| cpt\_codes | String (JSON) | Array of CPT codes with section headers and descriptions | "[{'CPT\_CD': '94003', 'SECTIONHEADER': 'Medicine', 'SUBSECTIONHEADER': 'Pulmonary', 'DESCRIPTION': 'VENT MGMT;SUBSQ DAYS(INVASIVE)'}...]" |

# Solution Design & Detailed Phases

**Phase 1: Data Preparation & Ingestion**

1. **Data Profiling & Cleaning** 
   * Inspect source\_records.csv for missing values, malformed JSON fields, duplicate records using Vertex AI Workbench
   * Standardize date formats (CHARTDATE), normalize whitespace, remove PHI if present
   * Tokenize or segment very long notes into 1-2 KB chunks for efficient embedding
2. **Data Storage Setup** 
   * Upload cleaned data to Cloud Storage buckets
   * Organize data for embedding processing pipeline

**Phase 2: Embedding Generation & Vector Indexing**

1. **Embedding Pipeline** 
   * Call Vertex AI Embeddings API to generate 1,024-dim vectors for each chunk
   * Attach metadata: row\_id, SUBJECT\_ID, HADM\_ID, original text snippet
   * Store embeddings in Cloud Storage
2. **Matching Engine Index Creation** (Script already provided)
   * Configure a Vertex AI Matching Engine index with appropriate shard count and distance metric (cosine)
   * Upload embeddings and metadata via the Vertex AI Python client
3. **Validation & Benchmarking** 
   * Run sample nearest-neighbor queries to verify sub-200 ms response times using Vertex AI Workbench
   * Compare returned row\_id against known similar examples to spot-check accuracy

**Phase 3: RAG Retrieval & Aggregation Agent**

1. **RetrieverAgent (LangGraph)**
   * Input: new clinical note text
   * Tasks:
     + Embed text via Vertex AI Embeddings
     + Query Matching Engine for top-K (e.g., K = 5) similar records
     + Retrieve associated ICD-9 & CPT code arrays
2. **Aggregation Logic**
   * Implement weighted voting:
     + Weight each record’s codes by its normalized similarity score
     + Select codes whose cumulative weight exceeds a threshold (e.g., 60%)
   * Compute an overall confidence score (e.g., average of top-K similarities)
3. **Threshold Decision**
   * If confidence ≥ 0.75, proceed to response formatting
   * Else, trigger ReasonerAgent

**Phase 4: Fallback LLM Reasoning Agent**

1. **ReasonerAgent (LangGraph)**
   * Craft a few-shot prompt template that includes:
     + Patient note excerpt
     + 3–5 nearest examples (text + codes) from RetrieverAgent
     + Instructions for “Return up to 5 ICD-9 codes and 3 CPT codes with brief descriptions.”
   * Invoke Vertex AI’s Gemini 2.0 Flash model via the Vertex AI client
   * Parse the LLM’s output into structured code lists; validate format compliance
2. **Error Handling & Retry**
   * If the LLM response is malformed or times out, retry up to 2× with back-off
   * Log all fallback invocations and failures for analysis

**Phase 5: Workflow Composition & API Exposure**

1. **LangGraph Workflow**
   * Define a single graph that sequences: RetrieverAgent → conditional ReasonerAgent → ResponseFormatter
   * Use clear node naming and inline documentation for maintainability
2. **ResponseFormatter**
   * Merge codes and confidences into a JSON payload:

**{**

**"row\_id": 1234,**

**"assigned\_icd9": ["250.00", "401.9"],**

**"assigned\_cpt": ["99213"],**

**"confidence": 0.82,**

**"used\_fallback": false**

**}**

1. **REST API Service**
   * Wrap the workflow in a Python Flask or FastAPI app
   * Deploy to Cloud Run or Vertex AI Endpoint
   * Secure with IAM authentication and validate inbound tokens

**Phase 6: Containerization & Deployment**

1. **Cloud Functions Deployment**
   * Package LangGraph runtime and Python dependencies
   * Deploy API endpoints using Cloud Functions
2. **App Engine Frontend**
   * Deploy simple web interface using App Engine
   * Connect to Cloud Functions backend

**Phase 7: Observability & Monitoring**

1. **Prompt-Level Tracing (LangFuse)**
   * Instrument each Embedding call, Matching Engine query, and LLM invocation
   * Record metrics: latency, token counts, similarity scores, fallback invocation flags
2. **System & Custom Metrics**
   * Push key metrics (fallback\_rate, avg\_confidence, assignment\_latency, error\_count) to Cloud Monitoring’s Managed Prometheus
   * Configure dashboards in Grafana (hosted on GKE or Cloud Monitoring UI)
3. **Alerting**
   * Create Cloud Monitoring alert policies:
     + fallback\_rate > 10% over 1 hr
     + assignment\_latency > 1 s over 5 min
     + error\_count > 5 over 10 min
   * Route alerts to Slack or PagerDuty via webhook integrations

**Phase 8: UI Development & Stakeholder Demo**

1. **Web UI (React/Next.js)**
   * Build a simple interface for:
     + Single-note upload (file or text paste)
     + Batch upload via CSV
     + Display of retrieved examples, assigned codes, confidence, and fallback indicator
     + Feedback form for manual corrections
2. **Demo Preparation**
   * Prepare a step-by-step guide showing:
     + High-similarity scenario (no fallback)
     + Low-similarity scenario (fallback reasoning)
     + Real-time dashboard updates and alert generation

# **Expected Deliverables**

1. **Code Assets**
   * LangGraph workflow definitions
   * Python scripts: data ingestion, embedding loader, Cloud Functions API
   * Prompt templates for both RetrieverAgent and ReasonerAgent
   * App Engine application code
2. **Google Cloud Configuration**
   * Cloud Deployment Manager templates for all GCP resources
   * Cloud Functions deployment configurations
   * Vertex AI Matching Engine setup scripts
3. **Deployment Artifacts**
   * Cloud Functions URLs
   * App Engine application URLs
   * Cloud IAM policy configurations
4. **Monitoring & Alerts**
   * Cloud Monitoring dashboard configurations
   * Alert policy definitions
5. **Documentation**
   * Architecture diagrams
   * API documentation for the /ehr/encode endpoint
   * Setup and troubleshooting guides