NURSING HOME RAG DEMO

Complete Build Guide: From Data Download to Live Testing

Dify.ai · Supabase · Langfuse · CMS Open Data

Hybrid RAG
Vector + BM25 Keyword

SQL Queries
Structured Filtering

Observability
Langfuse Evaluation

Project Overview

This guide walks you through building a no-code AI demo that showcases the power of combining three retrieval modes: vector search, keyword search, and direct SQL queries. The domain is nursing home quality data from CMS — real, publicly available, emotionally resonant, and perfectly structured to demonstrate why hybrid search matters.

Architecture at a Glance

Component	Tool	Purpose	Cost
Data Source	CMS data.cms.gov	3 free CSVs, CC0 license	Free
Database	Supabase	PostgreSQL + REST API for SQL queries	Free tier
RAG + Workflow	Dify.ai	Hybrid search + workflow orchestration	Free tier
Observability	Langfuse	Tracing, evals, LLM monitoring	Free tier
LLM	Claude Sonnet / GPT-4o	Query routing + answer generation	Pay per use

The Three Query Types Your Demo Will Handle

Pure Semantic (Vector) Search

User asks: "Find facilities where residents felt emotionally neglected"
→ No exact keyword exists for this concept. The LLM embeds the query and finds semantically similar passages in inspector narratives.

Pure Keyword (BM25) Search

User asks: "Find deficiency reports mentioning elopement or wandering"
→ These are precise clinical terms. BM25 finds exact matches that semantic search might dilute or miss entirely.

Pure SQL Query

User asks: "Show me 5-star facilities in Florida with fewer than 3 penalties"
→ Pure structured filtering. No text search needed — the LLM writes SQL against your Supabase tables.

Hybrid Query (Your Best Demo Moment)

User asks: "Find nursing homes in California with memory care violations, rated below 3 stars, with more than 5 deficiencies in the last 2 years"
→ 'memory care violations' hits vector search over narratives
→ 'below 3 stars' and '5 deficiencies' hits SQL — impossible without both

Phase 1

Data Download & Preparation

~1-2 hours · No tools required

Step 1.1 — Download the Three CMS Datasets

Go to each URL below and click the CSV Download button. No account required.

Dataset	URL & What to Download
Provider Info (Facility metadata)	data.cms.gov/provider-data/dataset/4pq5-n9py Save as: providers.csv

Health Deficiencies (Narrative inspection text)	data.cms.gov/provider-data/dataset/r5ix-sfxw Save as: deficiencies.csv
Quality Measures (Numeric scores)	data.cms.gov/provider-data/dataset/djen-97ju Save as: quality_measures.csv

> **Tip**
> The Health Deficiencies dataset is the most important for your demo — it contains inspector narrative text like 'staff failed to reposition resident resulting in stage 3 pressure ulcer'. This is what makes vector search come alive.
>
> The common linking key across all 3 files is: Federal Provider Number

Step 1.2 — Understand the Key Columns

These are the columns you'll reference most in prompts and queries:

Column Name	Dataset & Description
federal_provider_number	All 3 datasets — the primary key linking everything
provder_name, city, state	providers.csv — facility identity and location
overall_rating (1-5)	providers.csv — the Five Star rating
total_amount_of_fines_in_dollars	providers.csv — financial penalties
deficiency_description	deficiencies.csv ← YOUR VECTOR SEARCH TARGET
deficiency_category	deficiencies.csv — category tag for keyword search
survey_date	deficiencies.csv — for date-based SQL filtering
measure_description	quality_measures.csv — human-readable measure name
score	quality_measures.csv — facility's score on that measure
national_average	quality_measures.csv — for comparison SQL queries

Phase 2

Supabase Setup

~1 hour · Database for SQL queries

Step 2.1 — Create Your Supabase Account

1. Go to supabase.com and click Start for free
2. Sign up with GitHub (recommended) or email

3. Click New Project — name it nursing-home-demo
4. Choose a region closest to you, set a database password, save it
5. Wait ~2 minutes for the project to spin up

Step 2.2 — Import the CSV Files

Supabase has a built-in CSV importer. For each of your 3 files:

6. In the left sidebar, click Table Editor
7. Click New Table → name it providers (then repeat for deficiencies and quality_measures)
8. Click Import data from CSV
9. Upload your CSV file — Supabase will auto-detect columns and types
10. Click Save

> ⚠ **Important — Fix Column Types**
> After importing, verify these column types in the Table Editor:
> • overall_rating → int4 (integer)
> • score, national_average → float8 (decimal)
> • survey_date → date
> • total_amount_of_fines_in_dollars → float8
> • federal_provider_number → text (NOT integer — it has leading zeros)
> Click the column header to edit type if needed.

Step 2.3 — Get Your Supabase API Credentials

11. In the left sidebar, click Project Settings → API
12. Copy and save these two values (you'll need them in Dify):

- Project URL — looks like: https://abcdefgh.supabase.co
- anon public key — a long JWT string starting with eyJ...

Step 2.4 — Enable the REST API for Dify

Supabase auto-generates a REST API for every table. Test it works:

```
# Open your browser and visit this URL (replace with your values):

https://YOUR_PROJECT_ID.supabase.co/rest/v1/providers?select=*&limit=3

# Add this header when testing with a tool like Postman or Insomnia:
```

```
# apikey: YOUR_ANON_KEY

# You should see JSON rows returned from your providers table
```

Phase 3

Dify.ai Setup & Knowledge Base

~2 hours · The RAG + workflow engine

Step 3.1 — Create Your Dify Account

13. Go to dify.ai and click Get Started
14. Sign up and log in
15. You'll land on the Studio dashboard

Step 3.2 — Create the Knowledge Base (Hybrid Search)

This is where you set up vector + keyword search over the deficiency narratives.

16. In the top navigation, click Knowledge
17. Click Create Knowledge → name it Nursing Home Deficiencies
18. Click Upload Files → upload your deficiencies.csv
19. In Indexing Mode, select High Quality (uses embeddings)
20. In Retrieval Settings, select Hybrid Search — this enables both vector and BM25 simultaneously
21. Set Chunk Size to 500 tokens, Overlap to 50
22. Click Save and Process — Dify will embed all narratives (takes 5-15 min)

> **Why These Chunk Settings**
>
> 500 tokens is roughly 2-3 inspection narrative paragraphs — enough context without being too broad. The 50-token overlap ensures no narrative gets cut off at the boundary between chunks, preserving the full context of each violation description.

Step 3.3 — Add Supabase as a Custom Tool

This lets your Dify workflow call SQL queries via the Supabase REST API.

23. Click your avatar (top right) → Settings → Tools → Custom Tools
24. Click Add Custom Tool

25. Name it: Supabase SQL Tool
26. In Schema, paste the OpenAPI spec below and replace YOUR_PROJECT_ID:

```
openapi: 3.0.0
info:
  title: Supabase Nursing Home Query
  version: 1.0.0
servers:
  - url: https://YOUR_PROJECT_ID.supabase.co/rest/v1
paths:
  /providers:
    get:
      operationId: query_providers
      summary: Query nursing home facility data
      parameters:
        - name: select
          in: query
          schema: { type: string }
        - name: state
          in: query
          schema: { type: string }
        - name: overall_rating
          in: query
          schema: { type: string }
        - name: limit
          in: query
          schema: { type: integer }
      responses:
        '200': { description: Array of nursing home facilities }
  /deficiencies:
    get:
      operationId: query_deficiencies
      summary: Query inspection deficiency records
      parameters:
        - name: select
          in: query
          schema: { type: string }
        - name: federal_provider_number
          in: query
          schema: { type: string }
        - name: limit
          in: query
          schema: { type: integer }
      responses:
        '200': { description: Array of deficiency records }
```

27. In Authentication, select API Key → add your Supabase anon key as the header: apikey
28. Click Test → it should return JSON from your providers table
29. Click Save

Phase 4

Build the Dify Workflow

~2-3 hours · The core AI logic

Step 4.1 — Create a New Chatflow

30. From the Dify dashboard, click Studio → Create App
31. Select Chatflow (not Agent or Completion)
32. Name it: Nursing Home Assistant
33. Click Create

Step 4.2 — The System Prompt (Schema Grounding)

This is the most important configuration in the entire project. Click the system prompt field and paste exactly this:

```
You are an expert assistant helping users search and analyze US nursing home data.
You have access to three retrieval methods and must choose the right one:

1. KNOWLEDGE BASE (Hybrid Search) — use for:
   - Conceptual or qualitative questions about care quality
   - Finding specific types of violations or incidents
   - Questions using descriptive language like 'emotional neglect', 'unsafe
conditions'

2. SUPABASE SQL TOOL — use for:
   - Filtering by state, city, star rating, penalty counts, dates
   - Counting, ranking, or comparing facilities numerically
   - Any question with numbers, thresholds, or geographic filters

3. BOTH — use for hybrid queries that combine qualitative concepts with numeric
filters

DATABASE SCHEMA (for SQL queries):

TABLE: providers
  federal_provider_number TEXT (primary key)
  provider_name TEXT
  city TEXT
  state TEXT (2-letter code, e.g. 'FL', 'CA')
  overall_rating INTEGER (1-5, Five Star rating)
  health_inspection_rating INTEGER (1-5)
  staffing_rating INTEGER (1-5)
  total_amount_of_fines_in_dollars FLOAT
  number_of_facility_reported_incidents INTEGER
```

```
TABLE: deficiencies
  federal_provider_number TEXT (foreign key → providers)
  deficiency_description TEXT (inspector narrative — searched via Knowledge Base)
  deficiency_category TEXT
  scope_severity_code TEXT
  survey_date DATE

TABLE: quality_measures
  federal_provider_number TEXT (foreign key → providers)
  measure_description TEXT
  score FLOAT
  national_average FLOAT

SUPABASE API RULES:
- Use filter syntax: column=eq.VALUE for equals, column=gt.VALUE for greater than,
  column=lt.VALUE for less than, column=like.*TERM* for text search
- Always include select=* or list specific columns
- Always add limit=20 unless the user asks for more

RESPONSE FORMAT:
- Always cite which retrieval method(s) you used
- Explain the facility name, location, and key finding for each result
- If combining methods, clearly show what each method contributed
- If no results found, explain why and suggest a refined query
```

Step 4.3 — Build the Workflow Nodes

In the workflow canvas, add and connect these nodes in order:

1 **Start Node**

Already exists. No changes needed.

2 **LLM Node — Query Classifier**

Add an LLM node. Model: Claude Sonnet or GPT-4o. Prompt:

`"Classify this user query. Output ONLY one of: SQL, VECTOR, HYBRID Query: {{#sys.query#}}"`

Set max tokens to 10. This is just a classifier — keep it lean.

3 **Condition Node — Route the Query**

Add an IF/ELSE node with 3 branches:

- Branch A: output contains 'SQL' → connect to Supabase Tool node
- Branch B: output contains 'VECTOR' → connect to Knowledge Retrieval node
- Branch C: output contains 'HYBRID' → connect to BOTH nodes in parallel

4 **Knowledge Retrieval Node**

Add a Knowledge Retrieval node:
- Knowledge Base: select 'Nursing Home Deficiencies'
- Query variable: `{{#sys.query#}}`
- Top K: 5 (returns 5 most relevant chunks)
- Score threshold: 0.5 (filters low-quality matches)

5 **Supabase Tool Node**

Add a Tool node → select 'Supabase SQL Tool':
- This node is called by an LLM that generates the API parameters
- Wrap it in an Agent node so the LLM can decide which endpoint to call
- The system prompt's schema grounding guides parameter generation

6 **Final LLM Node — Answer Synthesizer**

Add a final LLM node that receives outputs from all branches. Prompt:

```
"User question: {{#sys.query#}}  Knowledge base results: {{#knowledge.result#}}  SQL results: {{#tool.output#}}  Synthesize a clear, helpful answer. Cite which retrieval method(s) you used and why."
```

Phase **5**

Langfuse Integration

~45 minutes · Observability & tracing

Step 5.1 — Create Your Langfuse Account

34. Go to langfuse.com → Sign up for free
35. Create a new project: nursing-home-demo
36. Go to Settings → API Keys → Create new API key pair
37. Save your: Public Key and Secret Key

Step 5.2 — Connect Langfuse to Dify

Dify has native Langfuse integration built in — no code required.

38. In Dify, click your avatar → Settings → Monitoring
39. Click Add monitoring → select Langfuse
40. Fill in the fields:

- Host: https://cloud.langfuse.com (or your self-hosted URL)
- Public Key: paste from Langfuse

- Secret Key: paste from Langfuse

41. Click Save and Test Connection — you should see a green checkmark
42. Enable monitoring and click Save

What Gets Traced Automatically

Once connected, every conversation in your Dify app will be traced in Langfuse:
• Full LLM input/output for every node
• Token usage and cost per node
• Latency for each step (classifier, retrieval, synthesis)
• Which retrieval branch was triggered
• The final answer and any errors

Step 5.3 — Add Custom Metadata for Better Tracing

In your classifier LLM node in Dify, add this to the metadata field so Langfuse can tag traces by retrieval type:

```
# In Dify node metadata (JSON format):
{
  "project": "nursing-home-demo",
  "retrieval_type": "{{#classifier.output#}}",
  "user_query_length": "{{#sys.query.length#}}"
}
```

Step 5.4 — Set Up Langfuse Dashboards

In Langfuse, navigate to your project and configure:

43. Traces view — see every conversation with full node-by-node breakdown
44. Metrics → create a chart for: Average latency by retrieval_type tag
45. Metrics → create a chart for: Token cost per query over time
46. Sessions → group traces by conversation to see multi-turn patterns

Phase 6

Evaluation Framework

How to measure if your demo actually works

What You're Evaluating

Your demo has three systems to evaluate independently and together: the query classifier, each retrieval mode, and the final synthesized answer. Here is the complete evaluation plan:

Step 6.1 — Build Your Golden Test Set

Create a spreadsheet with at least 15 test queries — 5 per retrieval type. This is your ground truth.

Query	Expected Type	Expected Result Contains	Pass Criteria
Find facilities in TX with overall_rating < 2	SQL	Texas facilities, low ratings	Returns structured list
Show CA nursing homes with fines > $50,000	SQL	California, fine amounts	Correct SQL filter applied
Which states have most deficiencies in 2023?	SQL	State ranking with counts	Aggregation correct
Find reports mentioning resident falls	VECTOR	Fall-related narratives	Semantically relevant chunks
Facilities with emotional neglect issues	VECTOR	Neglect-related narratives	Captures concept not keyword
Poor hygiene or unsanitary conditions found	VECTOR	Hygiene violation narratives	Finds related language
Find elopement incidents in 2023	KEYWORD	Elopement exact matches	Exact term found
Reports citing F-tag F758	KEYWORD	F758 regulatory citations	Exact code matched
Memory care facilities below 3 stars in FL	HYBRID	FL + low rating + memory care	Both SQL + vector used
Understaffed CA homes with abuse reports	HYBRID	CA + staffing + abuse narratives	Both retrievals combined

Step 6.2 — Classifier Accuracy Evaluation

Run each test query through your system and record what the classifier returned vs. what you expected.

```
Target metric: > 85% classification accuracy
```

```
How to measure in Langfuse:
1. Go to Traces → filter by your 15 test queries
2. For each trace, look at the classifier node output
3. Compare to your golden test set 'Expected Type' column
4. Calculate: Correct Classifications / Total Queries

If accuracy < 85%, improve the classifier prompt by adding
more examples of each query type as few-shot examples.
```

Step 6.3 — Retrieval Quality Evaluation

For vector/hybrid queries, evaluate whether the retrieved chunks are actually relevant.

Metric	How to Measure
Relevance@K	For each VECTOR query, do the top 5 returned chunks actually relate to the question? Score 1-5 manually.
Precision	Of the 5 returned chunks, how many are truly relevant? Target: >3/5
Recall proxy	Does the answer contain the key facts you'd expect? Check against 3 known deficiency reports.
Hybrid vs. Vector only	Run same query in both modes, compare result quality manually
SQL correctness	For SQL queries, check if returned facilities actually match the filter criteria

Step 6.4 — Answer Quality Evaluation (with Langfuse Scores)

Langfuse lets you add evaluation scores to any trace. Use this to score your synthesized answers:

47. In Langfuse, open any trace → click Add Score
48. Create these score types in your project settings:

Score Name	Scale & What It Measures
faithfulness	0-1 · Does the answer only use retrieved content? (no hallucination)
relevance	0-1 · Does the answer actually address the user's question?
retrieval_mode_correct	0 or 1 · Did it use the right retrieval method?
sql_accuracy	0 or 1 · For SQL queries: did the SQL filter work correctly?
answer_completeness	0-1 · Did it capture all relevant findings from retrieved content?

Automated Evaluation with LLM-as-Judge

You can automate scoring using Langfuse's eval templates. Navigate to:
Langfuse → Evals → Create Eval Template → select 'Hallucination' or 'Relevance'
Langfuse will run an LLM judge against your traces automatically.

This turns manual spot-checking into a scalable automated pipeline.
Run it against all 15 golden test queries after every workflow change.

Step 6.5 — A/B Testing Retrieval Strategies

Use Langfuse's experiment tracking to compare different configurations:

49. In Dify, duplicate your chatflow → name it v2
50. Change one variable (e.g., Top K from 5 to 10, or chunk size from 500 to 1000)
51. Run your 15 golden queries against both versions
52. In Langfuse, filter traces by app version tag and compare average scores
53. Keep the configuration with better faithfulness + relevance scores

Step 6.6 — Key Metrics Dashboard (What to Show Stakeholders)

Metric	Target	Where to Find	Why It Matters
Classifier accuracy	> 85%	Manual check vs. golden set	Wrong routing kills the demo
Vector relevance@5	> 3/5 relevant	Manual scoring in Langfuse	Core RAG quality
SQL correctness	> 90%	Manual result verification	Structured query reliability
Faithfulness score	> 0.8	Langfuse LLM-as-judge	No hallucination
Avg. latency	< 8 seconds	Langfuse Metrics tab	User experience
Cost per query	Track trend	Langfuse token tracking	Production readiness

Phase 7

Testing Your Demo

~1 hour · Go live and validate

Step 7.1 — Publish Your Dify App

54. In Dify, click Publish → Web App
55. Toggle the public access switch → copy the shareable URL
56. Open the URL in a new browser tab — this is your demo interface

Step 7.2 — The Demo Script

Use these 8 queries in order during a demo — they build from simple to complex:

Demo Query 1 — Pure SQL (warm up)
"Show me all 5-star nursing homes in Texas"
Expected: Structured list from providers table, clearly a SQL call

Demo Query 2 — Pure SQL with math
"Which state has the highest average number of fines?"
Expected: LLM generates aggregation SQL, returns ranked state list

Demo Query 3 — Pure Keyword
"Find inspection reports that mention elopement"
Expected: BM25 exact match, returns specific deficiency records

Demo Query 4 — Pure Vector (the wow moment)
"Find facilities where residents felt like they had no dignity"
Expected: Semantic search finds emotional abuse narratives — no exact keyword match

Demo Query 5 — Vector shows its weakness
"Find reports mentioning F-tag F684"
Expected: Vector may miss or dilute this — shows why keyword matters too

Demo Query 6 — Hybrid (the main event)
"Find nursing homes in California with memory care problems, rated below 3 stars"
Expected: Vector finds memory care narratives, SQL filters by state + rating. Show both results.

Demo Query 7 — Complex Hybrid

"What are the most common safety violations in Florida's lowest-rated facilities?"
Expected: SQL identifies low-rated FL facilities, vector retrieves their violation narratives, LLM synthesizes themes

Demo Query 8 — Live Langfuse walkthrough

After running query 7, open Langfuse and show the audience the trace:
→ Classifier node output, retrieval results, token cost, latency per node
→ This is your observability story

Step 7.3 — Common Issues & Fixes

Problem	Fix
Classifier always returns VECTOR	Add few-shot examples to classifier prompt showing SQL and HYBRID queries
SQL queries return no results	Check Supabase column names match your system prompt schema exactly
Vector results are irrelevant	Lower the score threshold from 0.5 to 0.3, or increase Top K to 8
Langfuse not receiving traces	Re-check the API keys in Dify Settings → Monitoring, ensure no trailing spaces
Hybrid queries only use one method	Verify the HYBRID branch in your condition node connects to BOTH retrieval nodes
Answers hallucinate facility names	Add to system prompt: 'Only mention facilities that appear in retrieved results'

Quick Reference Summary

Phase	What You Do	Time	Output
1 — Data	Download 3 CMS CSVs	30 min	providers.csv, deficiencies.csv, quality_measures.csv
2 — Supabase	Import CSVs, get API credentials	60 min	Live PostgreSQL DB with REST API
3 — Dify KB	Upload deficiencies, enable hybrid search	60 min	Indexed knowledge base with vector+BM25

4 — Workflow	Build classifier + retrieval + synthesis nodes	2-3 hrs	Working chatflow with 3 retrieval modes
5 — Langfuse	Connect monitoring, add trace metadata	45 min	Full observability dashboard
6 — Evaluation	Run golden test set, score with LLM judge	2 hrs	Accuracy metrics + A/B test results
7 — Testing	Publish, run demo script, fix issues	60 min	Shareable demo URL

Total Estimated Time

First build: 8-12 hours across 2-3 days
After initial setup: 30 minutes to make changes and re-test
No code required at any step — everything is visual or configuration-based

Biggest time investment: Phase 4 (workflow building) and Phase 6 (evaluation)
Easiest step: Phase 5 (Langfuse) — Dify's native integration takes < 10 minutes