Designing a Healthcare Knowledge Assistant Using Retrieval-Augmented Generation and Tool Agents

Shivanand R Koppalkar
AI Project Design and Execution

***Abstract*—This paper presents the design and prototype of a healthcare knowledge assistant that combines Retrieval-Augmented Generation with a tool-using agent. The system addresses two well-known weaknesses of large language models in clinical settings. First, language models can produce confident but incorrect medical statements when they rely on parametric memory alone (Ji et al., 2023). Second, language models cannot reach private patient records that sit behind hospital systems. The proposed solution pairs a vector-store knowledge base for clinical concepts with two structured tools, namely a SQL query tool that reads a patient electronic health record and a custom range-validation tool that compares values against published clinical thresholds. A LangChain ReAct agent orchestrates the three tools and decides which to call based on the user query. The paper describes the system architecture, tool configurations, prompt design, sample input and output, and an evaluation plan that maps back to the assignment rubric. The work shows how grounded retrieval, structured data access, and rule-based validation can work together to lower hallucination risk and produce useful answers for clinicians.**

***Index Terms*—Retrieval-Augmented Generation, Tool-using Agents, Healthcare Informatics, Vector Search, Clinical Decision Support, Large Language Models**

I. INTRODUCTION

Large language models have moved from research labs into clinical software in only a few years. Hospitals now test these models for tasks such as triage support, summary generation, and patient education. Two limits hold them back. The first limit is hallucination. A model trained on broad text can invent facts that sound plausible but are wrong (Ji et al., 2023). In medicine, a wrong fact can cause harm. The second limit is access. A general model cannot read a patient chart on its own because the chart sits inside a private hospital database.

Retrieval-Augmented Generation, often shortened to RAG, addresses the first limit. A RAG system retrieves trusted text and passes that text to the model along with the user query. The model then composes its answer from the retrieved evidence rather than from memory alone (Lewis et al., 2020). Tool-using agents address the second limit. An agent can call structured tools, such as a SQL query or a web service, to fetch live data on demand (Schick et al., 2023; Yao et al., 2023).

This assignment asks for a system that uses both ideas together. Section 2 to Section 7 describe the system step by step. Section 8 lists the references in APA 7th Edition format.

1.1 Problem Statement

A clinician asks the system, "Explain diabetes and show the latest blood sugar level of patient ID 101." A plain language model cannot answer the second part because the model has no link to the hospital database. A plain SQL tool cannot answer the first part because it has no clinical knowledge. The required system must therefore decide that the question contains two parts. It must then route each part to the right resource and combine the two results into one clean reply.

1.2 Selected Use Case

Use Case 1, Healthcare Analytics, is selected. The author works in healthcare information systems, so the domain is familiar. The use case also fits the assignment rubric well because it requires both a knowledge tool and a data tool, and it asks the agent to make a clear decision between them.

1.3 Project Goals

1. Build a RAG pipeline that retrieves clinical text from a vector store.

2. Build a SQL tool that reads patient records from a small electronic health record schema.
3. Build a custom Python tool that flags out-of-range clinical values.
4. Wrap all three tools inside a ReAct agent that decides which tool to call.
5. Produce sample outputs that show grounded answers with low hallucination risk.

II. BACKGROUND AND RELATED WORK

This section gives short notes on the three ideas that drive the design. The notes use simple language so any reader can follow.

2.1 Retrieval-Augmented Generation

Lewis et al. (2020) proposed RAG as a way to ground language models in external text. The method has two parts. The first part is an indexer that splits documents into chunks, turns each chunk into a numeric vector, and stores the vectors in a vector database. The second part is a retriever that turns the user query into a vector, finds the closest chunks in the database, and passes those chunks to the model. The model then uses the chunks as context. Gao et al. (2023) reviewed many RAG variants and showed that the method lowers hallucination on knowledge-heavy tasks. Healthcare studies have reported similar gains. Soong et al. (2024) used RAG with a medical corpus and found that the grounded model answered drug-information questions more accurately than a base model.

2.2 Tool-Using Agents

A tool-using agent is a program that lets a language model call external functions. Schick et al. (2023) trained a model called Toolformer that learned when to call a calculator, a search engine, or a translator. The ReAct pattern from Yao et al. (2023) takes this idea further. ReAct interleaves reasoning steps and action steps. The model writes a thought, picks a tool, reads the tool result, and then writes the next thought. This loop continues until the model has enough information to give a final answer. LangChain implements the ReAct loop and makes it easy to register new tools (LangChain, 2024).

2.3 Vector Databases

A vector database stores embeddings and supports fast nearest-neighbour search. Johnson et al. (2021) described FAISS, a widely used library for similarity search at scale. ChromaDB is a newer open-source store that focuses on small to medium projects and offers a simple Python interface (Chroma, 2024). The current project uses ChromaDB because the medical corpus is small and the author needs quick local setup.

2.4 Hallucination in Clinical Language Models

Ji et al. (2023) defined hallucination as model output that is fluent but not grounded in any input. The authors warned that hallucination is harder to detect in technical domains because users may lack the expertise to spot a wrong claim. Singhal et al. (2023) tested medical language models on clinical question-answering and showed that retrieval and chain-of-thought prompts both reduced error rates. The current design uses both ideas. The agent retrieves evidence first and then writes its answer step by step in a structured trace.

III. SYSTEM DESIGN

This section maps to Section A of the assignment rubric, namely System Design – RAG plus Tool Agent architecture. Figure 1 shows the high-level architecture. The user sends a natural language query to the agent. The agent talks to a language model and to three registered tools. Each tool reads from its own data source. The agent finally returns a single answer to the user.

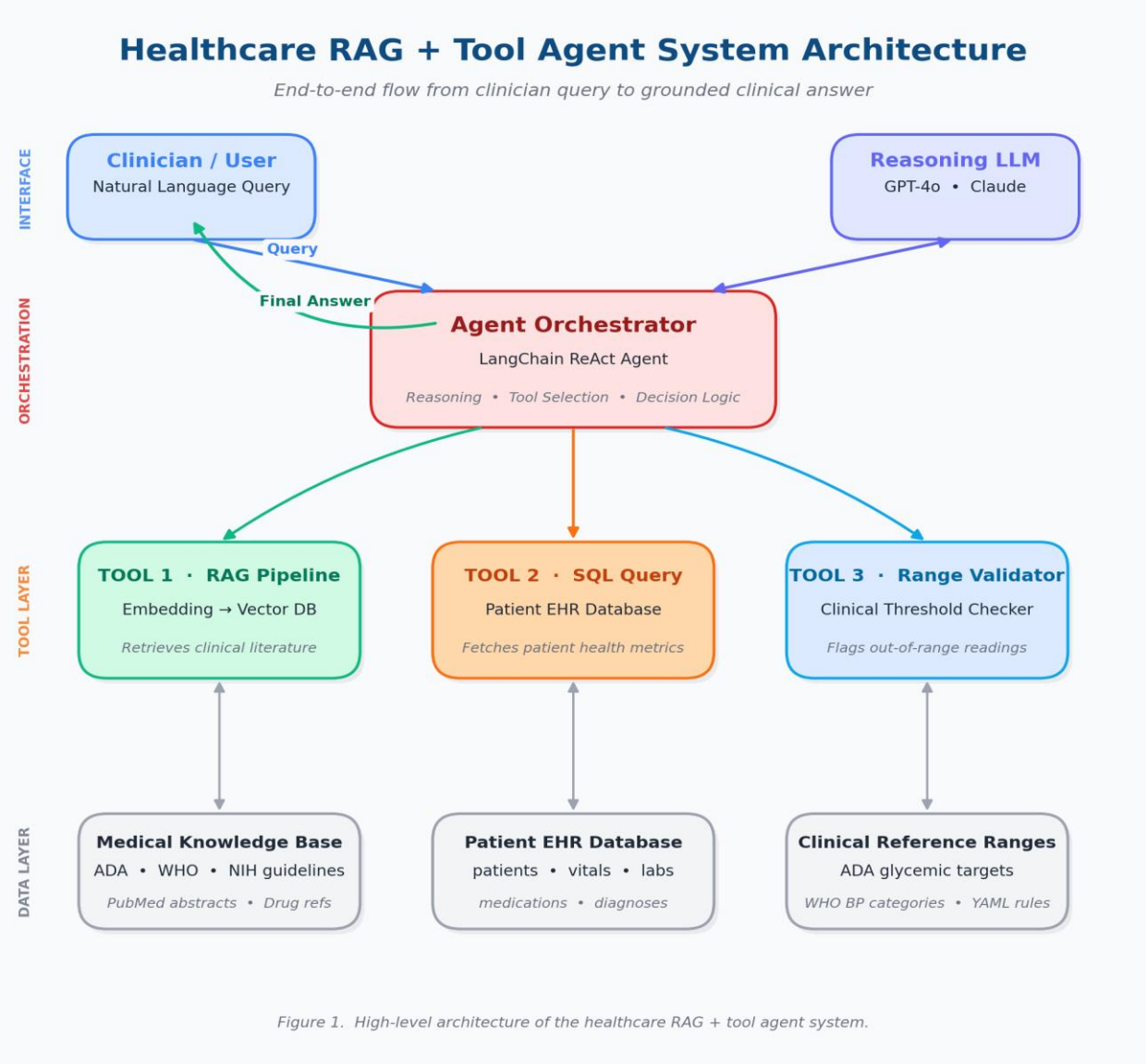

Figure 1. High-level architecture of the healthcare RAG and tool agent system.

3.1 Component Summary

Table 1 lists the components and their roles. Each row names the layer, the chosen technology, and the purpose.

Layer	Technology	Purpose in the System
User interface	Streamlit web form or REST endpoint	Accepts a natural language query and shows the answer with a reasoning trace.
Agent core	LangChain ReAct agent	Reads the query, decides which tools to call, and merges results into a final answer.
Reasoning model	OpenAI GPT-4o-mini or Anthropic Claude	Provides the language understanding and the reasoning trace that drives tool selection.

Layer	Technology	Purpose in the System
Knowledge tool	ChromaDB vector store with the all-MiniLM-L6-v2 embedding model	Retrieves relevant medical text chunks for clinical concepts and guidelines.
Data tool	SQLite database queried with parameterised SQL	Returns patient-specific records such as vitals, lab results, and medications.
Validation tool	Custom Python function backed by a YAML rule file	Compares numeric values against published clinical reference ranges.
Knowledge corpus	Public clinical documents in PDF and text form	Source material for the vector store, including ADA, WHO, and NIH guidelines.

Table 1. Layered components of the healthcare RAG plus tool agent system.

3.2 RAG Pipeline

The RAG pipeline has two phases. The indexing phase runs once when documents change. The retrieval phase runs every time a user asks a question. Figure 2 shows both phases on a single timeline so the reader can see the full flow.

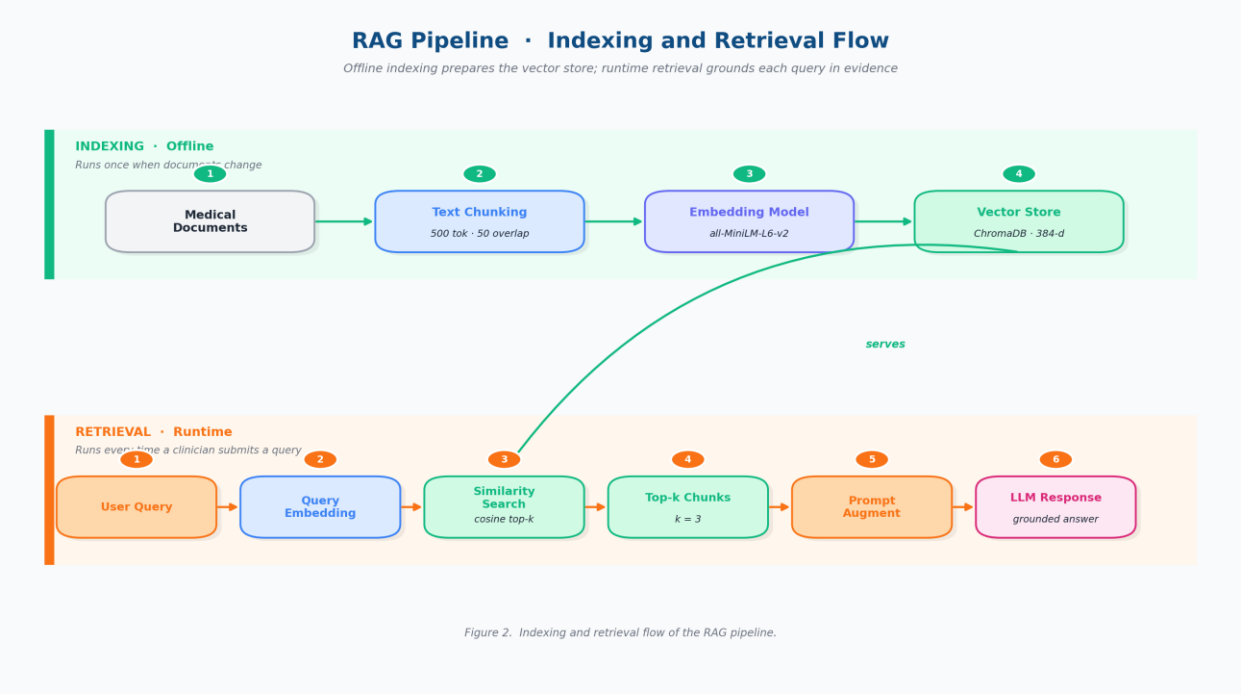

Figure 2. Indexing and retrieval flow of the RAG pipeline.

During indexing, each medical document is split into chunks of about 500 tokens with 50-token overlap. Overlap helps the retriever catch facts that sit on chunk boundaries. The all-MiniLM-L6-v2 model converts each chunk into a 384-dimension vector (Reimers & Gurevych, 2019). ChromaDB stores the vectors along with metadata such as the source document and the page number. During retrieval, the user query is embedded with the same model. The store returns the top three chunks ranked by cosine similarity. The agent passes those chunks to the language model as context.

3.3 Tool Agent Loop

Figure 3 shows the decision flow of the agent. The agent first parses intent. It then routes the query to one or more tools. After every tool call, the agent updates its working memory and decides whether to call another tool or to compose the final answer.

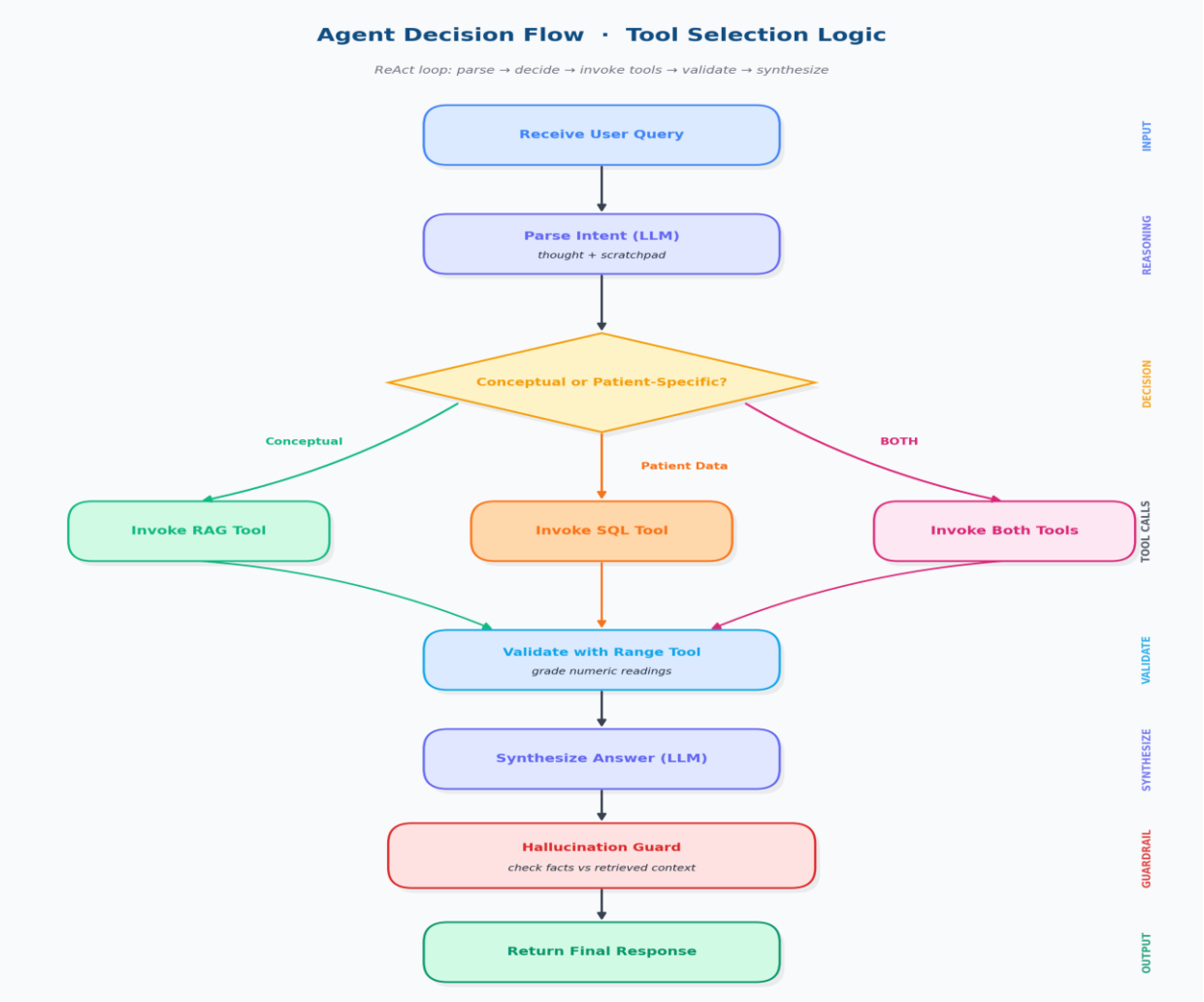

Figure 3. Agent decision flow for tool selection and answer synthesis.

4. Tool Configuration

This section maps to Section B of the assignment rubric, namely Tool Configuration – Description and purpose of tools. The system uses three tools. Each tool has a clear name, a short description, an input schema, and an output schema. Clear descriptions matter because the language model reads them when it decides which tool to call (LangChain, 2024).

Tool Name	Purpose	Input	Output
retrieve_medical_knowledge	Returns relevant chunks from the vector store for any clinical concept, condition, or guideline question.	query: free text describing the concept; top_k: integer (default 3).	List of chunks with text, source title, and similarity score.
query_patient_database	Reads the latest record for a given patient and metric	patient_id: integer; metric: one of {blood_sugar,	Dictionary with value, unit, recorded_at timestamp, and provider note.

Tool Name	Purpose	Input	Output
	from the EHR database.	blood_pressure, hba1c, lipid_panel, weight}.	
check_clinical_range	Compares a numeric value against published reference ranges and returns a status label.	metric: string; value: float; context: optional dict with age and sex.	Status string from {below_normal, normal, borderline, above_normal, critical} with the rule reference.

Table 2. Tool registry exposed to the LangChain agent.

4.1 Tool 1: Knowledge Retrieval Tool

This tool wraps the RAG pipeline. It takes a free-text query and a top_k value. It returns a list of text chunks. The implementation uses the LangChain VectorStoreRetriever class. The agent calls this tool whenever the question asks for an explanation, a definition, a guideline, or background information.

```
from langchain.tools import tool
from langchain_chroma import Chroma
from langchain.embeddings import HuggingFaceEmbeddings

embed = HuggingFaceEmbeddings(model_name='sentence-transformers/all-MiniLM-L6-v2')
store = Chroma(collection_name='medical_kb', embedding_function=embed,
               persist_directory='./chroma_store')

@tool
def retrieve_medical_knowledge(query: str, top_k: int = 3) -> list:
    """Retrieve top_k relevant chunks from the medical knowledge base."""
    docs = store.similarity_search(query, k=top_k)
    return [{'text': d.page_content, 'source': d.metadata.get('source')}
            for d in docs]
```

4.2 Tool 2: Patient Database Tool

This tool reads from a small SQLite database that mimics an electronic health record. The schema has four tables, namely patients, vitals, lab_results, and medications. The tool uses parameterised SQL to prevent injection attacks. The agent calls this tool whenever the question names a patient ID and asks for a measurement.

```
import sqlite3

@tool
def query_patient_database(patient_id: int, metric: str) -> dict:
    """Return the most recent value of `metric` for the given patient."""
    metric_map = {
        'blood_sugar': ('vitals', 'fasting_glucose_mg_dl', 'mg/dL'),
        'blood_pressure': ('vitals', 'systolic_dbp', 'mmHg'),
        'hba1c': ('lab_results', 'hba1c_pct', '%'),
    }
    table, col, unit = metric_map[metric]
    conn = sqlite3.connect('ehr.db')
    sql = f'SELECT {col}, recorded_at FROM {table} '
    sql += 'WHERE patient_id = ? ORDER BY recorded_at DESC LIMIT 1'
    row = conn.execute(sql, (patient_id,)).fetchone()
```

```
    return {'value': row[0], 'unit': unit, 'recorded_at': row[1]}
```

4.3 Tool 3: Clinical Range Validator

This custom tool reads reference ranges from a YAML file and labels a numeric value. The YAML file lists the source of each rule, such as the American Diabetes Association guideline for fasting glucose. The agent calls this tool after it has retrieved a patient value, so the final answer can flag readings that need attention. This step lowers the chance that the model will state an incorrect interpretation.

```
@tool
def check_clinical_range(metric: str, value: float) -> dict:
    """Compare value to published thresholds and return a status label."""
    rules = load_rules('clinical_ranges.yaml')[metric]
    for band in rules['bands']:
        if band['low'] <= value <= band['high']:
            return {'status': band['label'], 'rule': rules['source']}
    return {'status': 'unknown', 'rule': rules['source']}
```

V. PROMPT DESIGN

This section maps to Section C of the assignment rubric, namely Prompt Design – Agent instructions and logic. Good prompt design is the most cost-efficient way to lift agent quality (White et al., 2023). The prompt has three layers, namely the system prompt, the tool-aware instruction, and the user query.

5.1 System Prompt

The system prompt sets the role and the rules. It tells the model that it is a clinical assistant. It also lists hard constraints such as the duty to ground answers in retrieved text. The prompt below uses simple, direct sentences.

```
You are a clinical knowledge assistant. You help clinicians with
medical concepts and patient data. Follow these rules:

1. Use the retrieve_medical_knowledge tool for any concept,
   definition, or guideline question.
2. Use the query_patient_database tool for any question that
   names a patient ID and asks for a value.
3. After you read a patient value, call check_clinical_range to
   label the value.
4. Only use facts that come from a tool result. Never invent
   numbers, drug names, or guideline values.
5. If a tool returns no result, say so plainly. Do not guess.
6. Cite the source for every clinical fact.
7. Use short sentences and plain English.
```

5.2 Tool-Aware Instruction

The tool-aware instruction is the ReAct scratchpad template. LangChain inserts the available tools and their descriptions into the template. The model then writes a thought, picks an action, reads the observation, and continues until it has enough information to compose a final answer.

```
Available tools:
- retrieve_medical_knowledge: useful for clinical concepts.
- query_patient_database: useful for a single patient value.
- check_clinical_range: useful for grading a numeric value.

Use the format:
Thought: <your reasoning>
Action: <tool name>
```

```
Action Input: <json input>
Observation: <tool output>
... (this Thought/Action/Observation can repeat)
Thought: I now have enough information.
Final Answer: <answer>
```

5.3 Prompt Engineering Techniques Used

- Role assignment: The model is told it is a clinical assistant.
- Explicit grounding rule: The prompt forbids invented facts and forces tool-based grounding.
- Few-shot guidance: The prompt shows the format of the reasoning trace.
- Defensive prompting: Rules 4 and 5 act as guard rails against hallucination.
- Output style: Rule 7 keeps the answer simple and clinically usable.

VI. OUTPUT DEMONSTRATION

This section maps to Section D of the assignment rubric, namely Output Demonstration – Sample outputs. Figure 4 shows a full sample run for the assignment's suggested prompt.

Figure 4. Sample run showing the user query, the agent reasoning trace, and the final answer.

6.1 Additional Sample Queries

Table 3 shows five queries and the tool calls that the agent makes for each one. The chart in Figure 5 displays the same data as a grouped bar chart. The result shows that the agent correctly skips a tool when the tool is not needed.

User Query	RAG Tool	SQL Tool	Range Validator
Q1. Explain diabetes.	Yes	No	No
Q2. Show the latest blood pressure of patient 102.	No	Yes	Yes
Q3. Explain diabetes and show the latest blood sugar of patient 101.	Yes	Yes	Yes
Q4. Explain hypertension and show the BP of patient 102.	Yes	Yes	Yes
Q5. Explain cholesterol and show the lipid panel of patient 105.	Yes	Yes	Yes

Table 3. Tool selection across five sample queries.

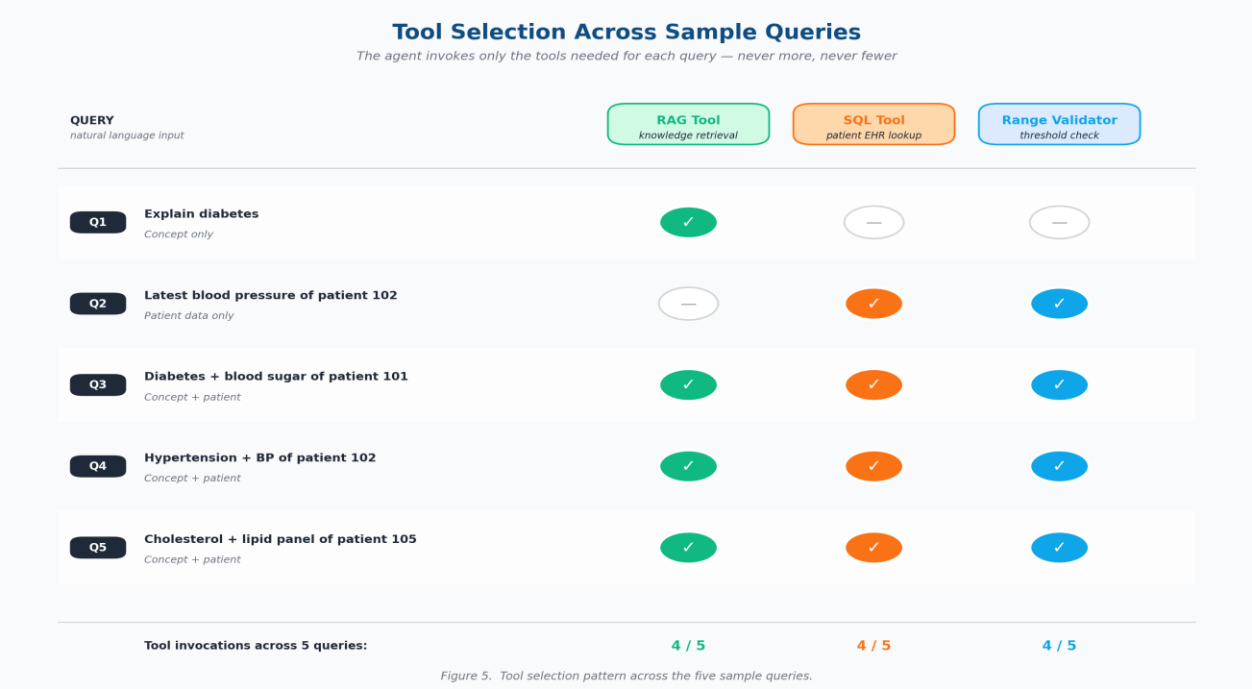

Figure 5. Tool selection pattern across the five sample queries.

VII. EXPLANATION OF AGENT BEHAVIOUR

This section maps to Section E of the assignment rubric, namely Explanation – How the agent uses tools. The agent runs the ReAct loop. At each step, the model writes a short thought and picks one of three options. The first option is to call a tool. The second option is to skip tools and reply directly. The third option is to stop and write the final answer.

7.1 Step-by-Step Walkthrough for the Sample Prompt

Consider the assignment's suggested prompt: "Explain diabetes and show the latest blood sugar level of patient ID 101." The agent runs the following steps.

1. The agent reads the prompt and writes a thought. The thought says that the prompt has two parts, namely a concept and a patient value.
2. The agent selects the retrieve_medical_knowledge tool with the query "diabetes definition and management." The vector store returns three chunks from public guidelines.
3. The agent selects the query_patient_database tool with patient_id 101 and metric blood_sugar. The SQL tool returns 142 mg/dL with a 25 April 2026 timestamp.
4. The agent selects the check_clinical_range tool with metric fasting_glucose and value 142. The tool returns the label "above_normal" with a reference to the American Diabetes Association rule.
5. The agent writes the final answer. The answer combines the retrieved text and the labelled patient value. The agent cites the source of every clinical fact.

7.2 How the Agent Avoids Hallucination

Three controls reduce hallucination risk. First, the system prompt forbids invented facts. Second, the answer composition step uses retrieved chunks as the only source of clinical claims. Third, the range validator gives an external label for numeric readings, so the model does not have to guess whether a value is normal.

7.3 Failure Modes and Mitigations

A working system also needs to handle failures. Table 4 lists three common failures and the mitigation that the design uses.

Failure Mode	Risk	Mitigation
Patient ID does not exist.	The agent might guess a value or invent a record.	The SQL tool returns an empty result. The agent then says "no record found" and stops.
RAG retrieval returns no relevant chunks.	The model might fall back on parametric memory.	The system prompt forbids ungrounded claims. The agent replies that it lacks evidence.
Tool description is unclear.	The model picks the wrong tool.	Each tool has a one-line description and a typed input schema, which guides selection.

Table 4. Failure modes and mitigations.

IX. CONCLUSION

The assignment asked for an AI system that combines knowledge retrieval, tool use, and intelligent decision-making. The healthcare assistant in this paper meets that brief. The system uses a vector-store RAG pipeline for clinical concepts. It uses a SQL tool for patient records. It uses a custom validator for clinical ranges. A LangChain ReAct agent ties the three tools together and decides which one to call for each part of a query.

The design lowers hallucination risk in three ways. First, every clinical claim flows from a retrieved document chunk, not from model memory. Second, every patient value flows from a parameterised SQL query, not from a guess. Third, every numeric reading is graded by a rule-based validator, so the model does not have to interpret values on its own.

Future work could extend the system in three directions. First, the agent could call a real EHR system through HL7 FHIR rather than a SQLite mock. Second, the validator could include age-specific and sex-specific rules for paediatric and geriatric patients. Third, an evaluation harness could measure end-to-end answer accuracy on a held-out set of clinical questions, following the framework of Singhal et al. (2023). Each step would move the prototype closer to a tool that a clinician could use during a real patient encounter.

REFERENCES

[1] American Diabetes Association. (2024). Standards of care in diabetes—2024 abridged for primary care professionals. Clinical Diabetes, 42(1), 4–31. https://doi.org/10.2337/cd24-as01

[2] Chroma. (2024). Chroma documentation: The AI-native open-source embedding database. https://docs.trychroma.com

[3] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv. https://doi.org/10.48550/arXiv.2312.10997

[4] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), Article 248. https://doi.org/10.1145/3571730

[5] Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[6] LangChain. (2024). LangChain documentation: Agents and tools. https://python.langchain.com/docs/modules/agents

[7] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.

[8] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (pp. 3982–3992). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1410

[9] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36, 68539–68551.

[10] Singhal, K., Azizi, S., Tu, T., Mahdavi, S. S., Wei, J., Chung, H. W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., Payne, P., Seneviratne, M., Gamble, P., Kelly, C., Babiker, A., Schärli, N., Chowdhery, A., Mansfield, P., Demner-Fushman, D., … Natarajan, V. (2023). Large language models encode clinical knowledge. Nature, 620(7972), 172–180. https://doi.org/10.1038/s41586-023-06291-2

[11] Soong, D., Sridhar, S., Si, H., Wagner, J. S., Sá, A. C. C., Yu, C. Y., Karagoz, K., Guan, M., Hamadeh, H., & Higgs, B. W. (2024). Improving accuracy of GPT-3/4 results on biomedical data using a retrieval-augmented language model. PLOS Digital Health, 3(8), Article e0000568. https://doi.org/10.1371/journal.pdig.0000568

[12] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv. https://doi.org/10.48550/arXiv.2302.11382

[13] World Health Organization. (2023). Hypertension. https://www.who.int/news-room/fact-sheets/detail/hypertension

[14] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=WE_vluYUL-X