🤖 Lhasa Generative AI POC Retrospective

Executive Summary

Executive Summary

Purpose of this document

This document captures key learnings and reflections on the Generative AI POC run by Endava and Lhasa from April to June 2025. Its purpose is to help Lhasa build on these learnings to plan further investment in GenAI, and to maximise the chances that these investments will lead to successful value creation for the business.

Goals of the POC

The primary goals of this project have been as follows:

- Enable knowledge transfer on foundational Generative AI topics for key Lhasa personnel
- Build learning on the specific capabilities of the AWS Bedrock platform and how it can support Lhasa's needs
- Deliver a working example of a GenAI application to build broader awareness of the capabilities of this technology within Lhasa
- Deliver insights about the practical implications of building and shipping GenAI capabilities within the Lhasa environment
- Enable further strategic decision-making about where and how GenAI approaches could support Lhasa's business
- Help to identify the roles, skills and operational/organisational model that will enable success for Lhasa in GenAI

High-level outcome

Key outcomes from this project have been as follows:

- Implemented and evaluated two techniques for retrieval-augmented generation: RAG Fusion and Agentic RAG
  - [results comparison]
- Successfully built an end-to-end framework for RAG-enabled natural language information retrieval
- Implemented both a deterministic and agentic query handling flow and compared the performance of the two of these
- Explored and evaluated the capabilities of Amazon Bedrock as a Generative AI inference platform and agentic platform
- Defined and tested an approach for automated and human-based testing of output
  - Created an automated test framework for this
- Learned about the characteristics of Vitic as a structured source of summarised toxicology data

Major conclusions and learnings

The major conclusions and learnings from this project are as follows:

- [list]

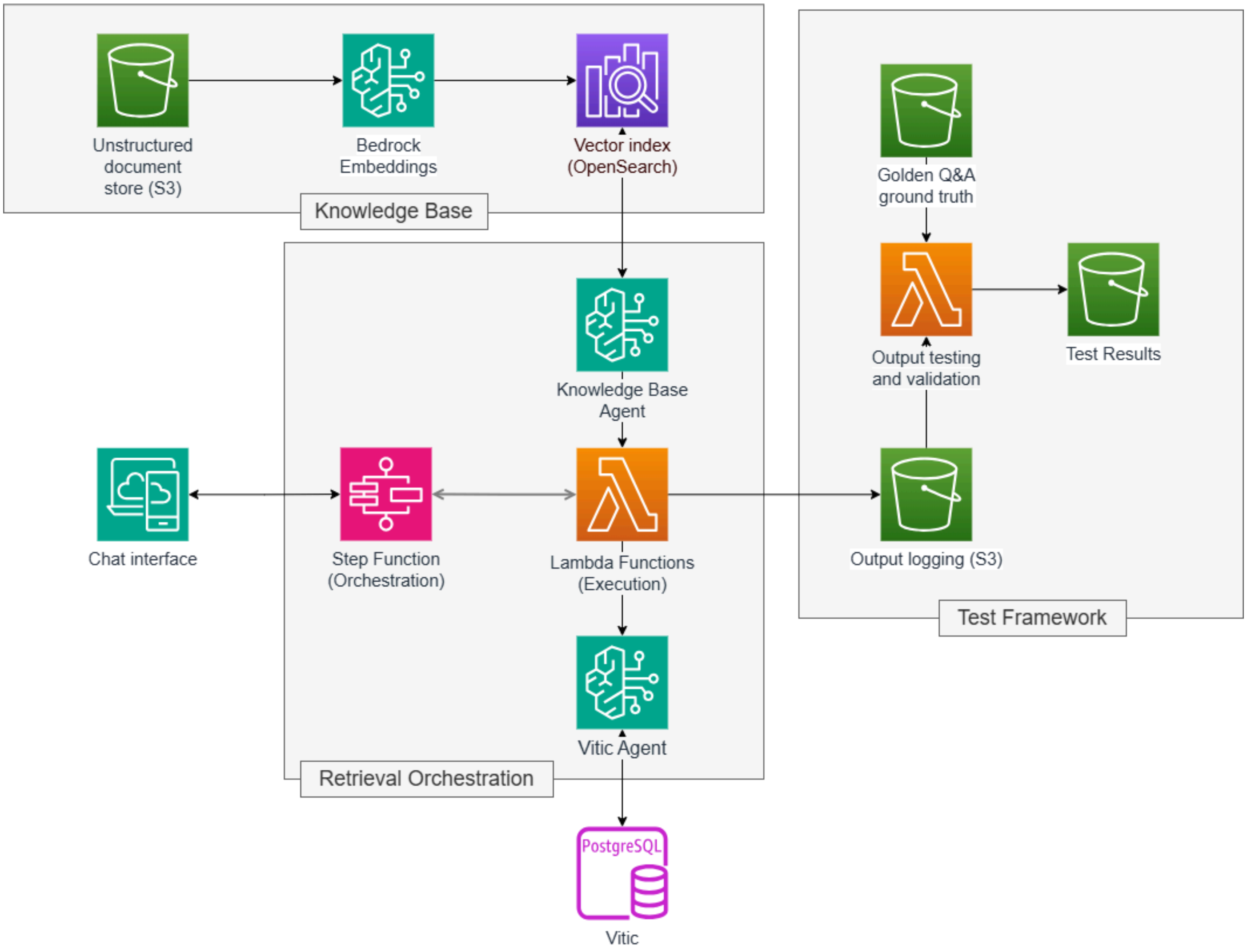
Architecture Overview

Architecture Overview

Note: This section can draw on / link to the existing documentation on the architecture

System Components and infrastructure

The overall high-level architecture of the system is as follows:



The architecture comprises four major components:

- A **Bedrock Knowledge Base**, created by indexing the unstructured document corpus provided in S3, with the resulting index held in AWS OpenSearch;
- A **Retrieval Orchestration** system, handling the invocation of multiple Bedrock model and agent instances to retrieve, combine and synthesize data from the Knowledge Base and from the Vitic database. This was built using a combination of Lambda functions, orchestrated with AWS Step Functions, supported by a combination of a set of Bedrock Agents and direct invocation of Bedrock models;
- A **Test Framework** that uses a set of “Golden Q&A” ground-truth data to assess whether the data retrieved from the Knowledge Base and Vitic successfully answers the original question asked; and
- A **Chat Interface** allowing users to input prompts to the system and perform manual assessments of the answers' quality and usefulness

Indexing Strategy

Knowledge Base indexing

Data sources

For the unstructured data, a number of data sources were prepared in advance, as follows:

- **cir** - Cosmetic Ingredient Review documents - 6551 pdf files, 648 unique
- **doi** - journal papers, pdf only - 345 pdf files

- **pubmed** - journal papers scraped from pubmed, 90% of content, pdf and metadata (json), ~70% of files with only abstract → for these files, we ingest both content and metadata - 135.3K metadata files, 33.55K pdf, 102K files abstract only txt files.
- **sccs** - Scientific Committee on Consumer Safety documents - 66 pdf files
- **zotero-papers** - Zotero is an reference management tool that can be used as an assistant for your research activity - 7648 pdf files

Data preprocessing

For each group of files, a different pre-processing was done:

- **cir** - eliminate duplicates using checksum and file dimension, copy - 648 pdf files
- **doi** - copy - 345 pdf files
- **pubmed** - extract abstract and metadata, creating metadata files, copy - 135.3 K metadata files, 135.5 K combined pdf & txt (abstract only) files
- **sccs** - copy - 66 pdf files
- **zotero-papers** - copy - 7628 pdf files

Retrieval Orchestration flows

During the lifetime of the POC project, most time was spent on building and optimising the retrieval orchestration part of the architecture, since this is the area where most impact could be made in improving the quality and utility of the answers generated. In particular, two broad approaches were used - a “RAG Fusion” approach and an “Agentic RAG” approach.

RAG Fusion retrieval

RAG Fusion is an advanced technique that enhances retrieval-augmented generation (RAG) solutions by combining information from multiple retrieval strategies before passing it to a generative language model. It was introduced in order to improve the relevance, diversity, and robustness of retrieved documents used in generative tasks like question answering.

With RAG, a question or prompt is used to retrieve documents from a knowledge base (based on a similarity search, a semantic search, a keyword search, or a combination of the last two). These documents are then passed to a generative language model to produce the final answer based on the initial query and the retrieved documents. The quality of the retrieved documents is critical to the quality of the generated response. The retrieved documents are very much dependent on the initial query. A possible problem with RAG is that the diversity of the documents retrieved is not always as expected. Also their relevance is sometime less than user would expect.

With RAG Fusion we introduce retrieval diversification and aggregation across multiple formulation of the same query. The steps of RAG Fusion are the following:

- Generate multiple query variants (e.g. by paraphrasing the original question)
- Retrieve top-k documents for each variant separately
- Aggregate all retrieved documents, by:
  - deduplication
  - reranking
  - weighted voting
- Pass the fused set of top-ranked documents into the generated model

We implemented a variant of the method, as following:

- Generate multiple query variants (e.g. by paraphrasing the original question)
- Run naive RAG for each query variant
- Pass the responses for each query to a Summariser Agent that will:
  - Summarise the responses
  - Deduplicate the references

The implementation of our RAG Fusion solution is described in Knowledge Base RAG Fusion Agent Confluence Page.

Agentic RAG retrieval

Agentic RAG extends traditional or naive RAG by allowing a model to:

- Planning - Formulate sub-questions or hypotheses
- Retrieval - Perform a retrieval using one of the sub-questions
- Analysis - Analyses the result of the retrieval
- Reflection - Reflect on the analysis of the retrieved information and adjust its behavior accordingly (e.g. generate a new sub-question for the next retrieval)
- Iterations with memory - Use memory, reasoning, and tool-use patterns over multiple turns (e.g. repeat the plan/retrieve/analyse/reflect cycle)

Ke principle of the Agentic RAG:

- Iterative Retrieval - the Agent will retrieve multiple times, possibly reformulating the query based on intermediate results
- Decomposition and Planning - for complex task, the agent might decompose the query into smaller steps
- Reasoning over Retrieved Evidence - retrieved documents are not just input - they are reasoned about, cross-references, and synthesised
- Memory and Working Context - the model retains intermediate information and uses it to refine the process (e.g. by deciding next query)
- Tools and Self Reflection - Agentic RAG might invoke retrieval tools, assess the usefulness of their outputs, and re-plan

Our implementation of Agentic RAG has 5 steps:

- **Planner** - invoke a **Bedrock Model** to generate current query; add the query to query history list; if route count > 0, will use for generating current query also the retrieval history
- **Retriever** - invoke a **Bedrock Agent** (**Knowledge Base**) to retrieve the current content (summary + citations) based on current query; add the retrieved data to the retrieval history
- **Analyser** - Use the current query, and the retrieval information from **Retriever Knowledge Base** and invoke a **Bedrock Model** to evaluate the quality of retrieval relative to the current query. Add a synthesis.
- **Reflector** - analyses the synthesis and the initial question and decides if the information retrieved in completed and provides a justification
- **Router** - decide based on routing number and decision if the next step is **Finalizer** or reroute to Planner. Condition is:
  - If verdict is COMPLETED or route_count >=3 will route to **Finalizer**
  - If verdict is INCOMPLETE and route_count < 3 will route to **Planner**
- **Finalizer** - summarises the information retrieved over all iterations - use a **Bedrock Model** to formulate the summary from the Knowledge Base summaries.

All details of the implementation are given in Agentic RAG Confluence page.

Agentic RAG - Multisource

This represents an extension of the Agentic RAG, and besides the unstructured data source, Vitic (SQL database) is used as source of information.

The retrieval and analysis for unstructured data (Knowledge Base) and Vitic (SQL database) are performed in paralel. The other steps (Planner, Reflector, Router, Finalizer) will run as before, with extended functionality to cover for the additional data source.

The steps are implemented as following:

- **Planner** - invoke a **Bedrock Model** to generate current query; add the query to query history list; if route count > 0, will use for generating current query also the retrieval history
- **Retriever Knowledge Base** - invoke a **Bedrock Agent** (**Knowledge Base**) to retrieve the current content (summary + citations) based on current query; add the retrieved data to the retrieval history
- **Retriever Vitic** - invoke a **Bedrock Agent** (**Vitic**) to retrieve the result of querying Vitic database with a SQL query generated based on current question and on the database schema. To retrieve the database schema, **Vitic Bedrock Agent** is using a function call,

`get_schema`. After the SQL query is generated, the **Vitic Bedrock Agent** executes the SQL query using a second function call, `run_schema`. Both function calls are available through one Action Group defined for the **Vitic Bedrock Agent**.

- **Analyzer KB** - use the current query, and the retrieval information from **Retriever Knowledge Base** and invoke a **Bedrock Model** to evaluate the quality of retrieval relative to the current query. Add a synthesis (`synthesis_kb`) (`synthesis_kb`).
- **Analyzer Vitic** - use the current query, and the retrieval information from **Retriever Vitic** and invoke a **Bedrock Model** to evaluate the quality of retrieval relative to the current query. Add a synthesis (`synthesis_sql`).
- **Reflector** - uses the two two synthesis (for KB & Vitic) and both both the retrieval histories (for KB & Vitic) and present them through invoking a **Bedrock Model** to provide a verdict and a justification.
- **Router** - decide based on routing number and decision if the next step is **Finalizer** or reroute to Planner. Condition is:
    - If verdict is COMPLETED or route_count >=3 will route to **Finalizer**
    - If verdict is INCOMPLETE and route_count < 3 will route to **Planner**
- **Finalizer** - Use a **Bedrock Model** to formulate the summary from the Knowledge Base summary & Vitic summary and reformat the input to prepare the output in a standardised form.

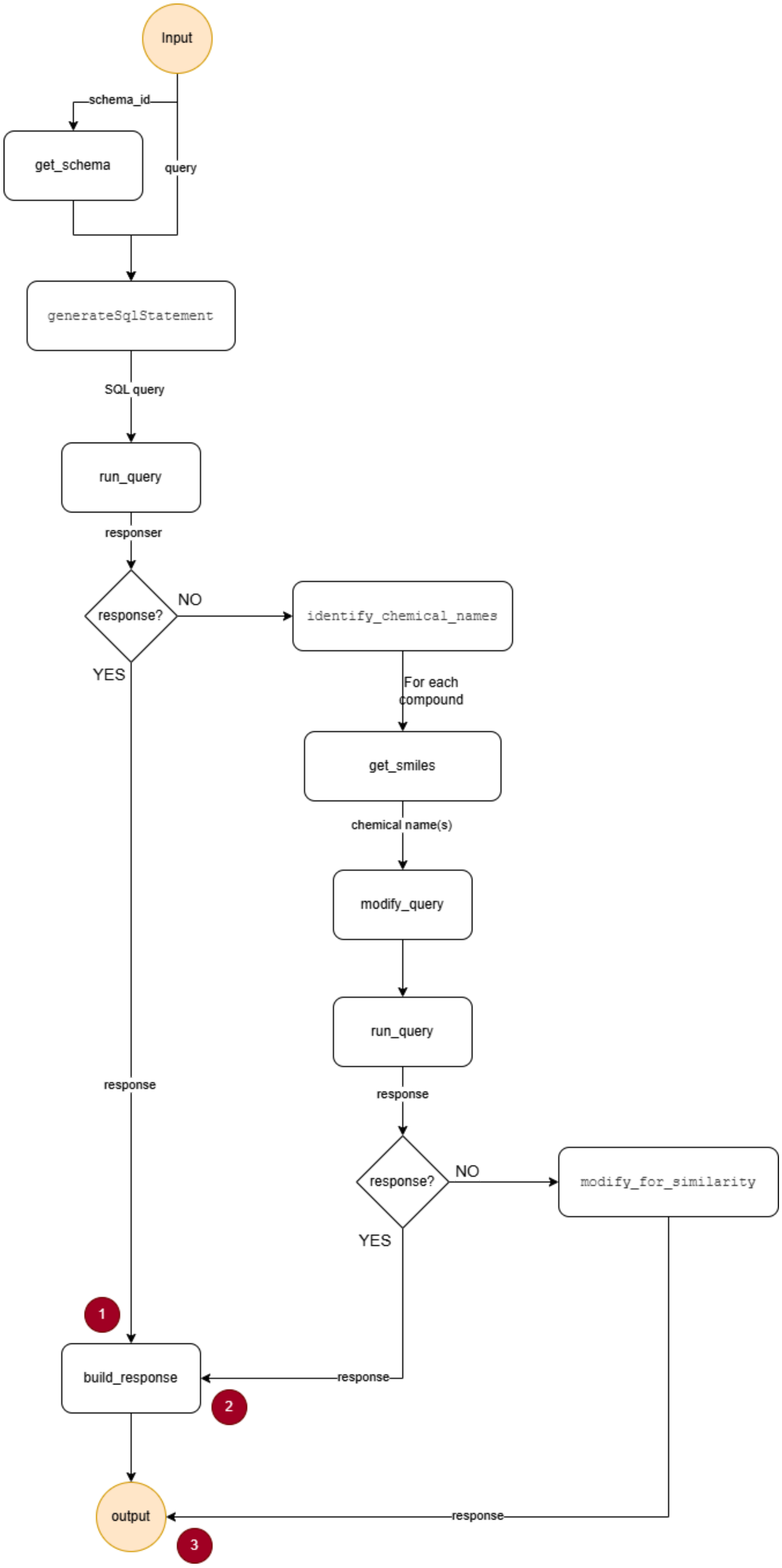
All details of the implementation are given in Agentic RAG - Multisource Confluence page.

Structured (Vitic) retrieval

@Orest Cramar

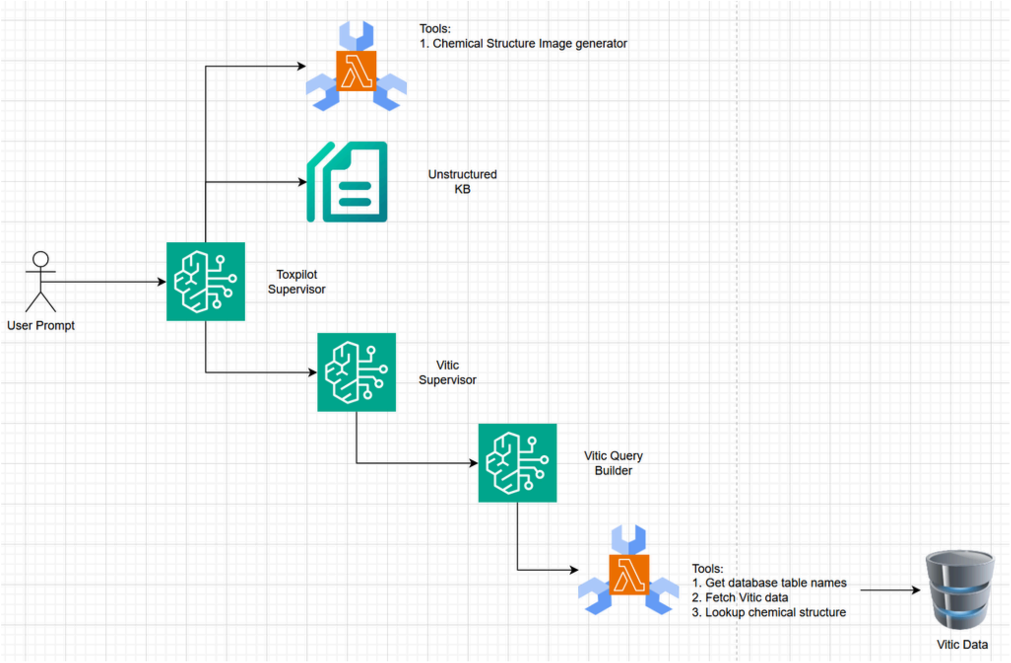
Vitic Agent is a SQL agent adapted to work with Vitic toxicology and carcinogenicity database. Four main implementations were created:

- Vitic Agent V1 - implementation in Vitic Agent V1
- Vitic Agent using Bedrock Agent with Action Group - one preliminary implementation in Vitic Agent with Action Group
- Vitic Agent V2 (see diagram below) - implementation in Vitic Agent V2


Input
schema_id
query
get_schema
generateSqlStatement
SQL query
run_query
responser
response?
NO
YES
identify_chemical_names
For each compound
get_smiles
chemical name(s)
modify_query
run_query
response
response?
NO
YES
modify_for_similarity
response
1
build_response
response
2
output
response
3

Agentic flow (Bedrock Agents)

As an alternative to the deterministic flows outlined above, and in order to explore further the native agentic orchestration capabilities of Bedrock, we created an agentic flow for prompt handling, as per the diagram below.



This approach delivered some useful learning and demonstrated the ability to use this mechanism to orchestrate query handling, though because of its agent-based orchestration it proved more challenging to constrain the systems behaviour, leading to the following issues:

- Low answer-to-question relevance
- Failure to directly answer questions, with irrelevant tangential information
- Over-explanation for simple queries
- Misinterpretation of Query Intent
- Hallucinations and Irrelevant Content

Test/evaluation framework

To evaluate the Knowledge Base (KB), we implemented a simple evaluation framework, based on the DeepEval framework and Python package. More information about this evaluation framework can be found in the following document:

Evaluation Framework for Knowledge Base

Development and testing process

Deploying the implementation

Information on deploying the POC implementation can be found here: Configuring Gen AI Chatbot Project

Deployment approach for testing

Test versions of the solution, primarily the retrieval orchestration components and associated system prompts, were deployed to the same AWS environment as Dev, using CDK. This lightweight approach meant that we did not have to incur the additional complexity of creating a full separate test environment with its own replicas of the Knowledge Base and Vitic database, but this meant that it was not possible to test changes to the indexing in the Knowledge Base, or changes to the structure of Vitic (though some additions were made to Vitic tables to facilitate retrieval).

A more robust testing approach would involve creating a full replica test environment with more rigorous version control, enabling the impact of feature changes to be assessed more clearly.

Human (UAT) testing

The creation of a simple web-based prompting interface enabled Lhasa scientists to test the system directly by entering their own prompts and assessing the results.

The results of this testing can be found here: https://lhasalimited.sharepoint.com/:x:/t/AIEndavaproject/EQ-urxK8JudDve4V9wwgp0YBNx7LX_VfcTMrZAZ7t5m_YQ?e=Nx7x4A Connect your OneDrive account

Testing summary

A summary of the testing outcomes can be found here: POC Testing Closure

Performance evaluation

To evaluate the performance of the system (in terms of the relevancy, accuracy and usefulness of the answers generated), we performed evaluation in the following three areas:

1. Assessing the similarity of the generated answer for a given question to a reference “golden” answer
2. Assessing the relevance and effective utilisation of retrieved data for the answer to a question
3. Human-judged testing of answer quality using a broader range of questions and answers

The first two of these areas were tested using a semi-automated evaluation framework generating a set of quantitative metrics. More information about this approach and the metrics can be found here: Evaluation Framework for Knowledge Base

Challenges, Learnings and Insights

Challenges, Learnings and Insights

AWS/Bedrock capabilities & limitations

Reasons to choose AWS/Bedrock

Amazon Bedrock was chosen as the inference and orchestration platform for this POC, for several reasons:

- Lhasa already provides several of its products (including Vitic) in an AWS hosted environment, facilitating integration
- There are good AWS deployment skills (via CDK) available within Lhasa
- Although a platform like OpenAI may have provided slightly more mature AI capabilities, there was a desire not to transit Lhasa proprietary data (either in terms of back-end integration or the actual data transfer involved in prompt response) outside of a Lhasa-managed environment
- Bedrock has quite a rich range of AI capabilities, including many leading models, plus agentic orchestration and facilitation of integrating unstructured data through its Knowledge Base features

AWS/Bedrock strengths

Bedrock proved to have several useful capabilities that made it easier to get up and running with the POC:

- **Knowledge base creation:** It is easy to use Bedrock via the console to create a Knowledge Base from a set of unstructured documents on s3. The setup process automates the setup of a Vector Store (in our case, AWS OpenSearch), and it is then easy to create Bedrock agents that can use the Knowledge Base to perform RAG query answering. However, using the Knowledge Base functionality does create some other limitations in terms of flexibility (see below).
- **Rich prototyping environment:** The Bedrock agent console experience, plus some other features like Flows and Prompt management, make it easy to prototype agents for particular tasks, and then either deploy these agents themselves or recreate workflows with Step Functions and Lambdas (as we did).
- **Agentic orchestration:** Bedrock agents can be configured to function as full AI agents, utilising external services and orchestrating the use of other agents using model-based reasoning. This creates the opportunity to create fully agentic teams of agents for complex

tasks, though controlling the behaviour of agents precisely can be a challenge, as we discovered during the POC.

- **Easy deployability with CDK:** Like other AWS services, Bedrock services are easy to deploy and manage with CDK.

AWS/Bedrock limitations

We encountered several limitations in working with Bedrock:

- **Regional variances in Bedrock capabilities:** We decided to implement the POC on the eu-west-2 (London) region, using Anthropic's Claude 3.0 models. However, this region does not receive the latest updates to Foundation Models as quickly as other regions (such as eu-west-1 or the US regions). Furthermore, Amazon has implemented Cross-Region Inference Services (CRIS) in these other regions, which is a feature that enables AWS to load-balance model calls across multiple sub-regions automatically, improving performance and mitigating against capacity caps; but this feature is not enabled in the London region.
- **Cross-region Bedrock limitations:** To address issues we encountered with capacity caps, and to enable access to more up-to-date models, we attempted to implement a copy of the system in the us-east-1 region (with CRIS enabled). However, we were not able to use the Knowledge Base from the London region in this other region, because the Bedrock agents we had created to leverage the Knowledge base cannot be configured to use a Knowledge Base outside of their region. There were several other instances of this kind of limitation with Bedrock agents - for example we were unable simply to update the agents already defined in the London region to use a US-hosted model for their inference.
- **Limited non-engineering test environment capabilities:** In order to enable UAT testing we needed a conversational interface that we could expose to scientists, and that they could use to provide feedback. AWS does not provide an easily configurable conversational UX that can be used for this purpose, forcing us to build our own. Amazon Q Business does provide some of this kind of functionality, but it is not available in the eu-west-2 region.

Vitic data

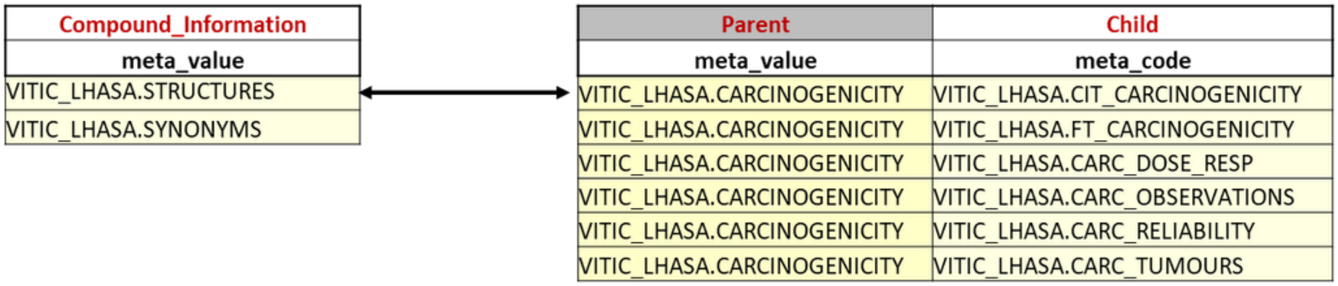
@Orest Cramar

For prompting against the PostgreSQL structured "Vitic" database we took an initial approach of feeding the database schema (described here: Vitic Metadata) for all the tables in the DB in the system prompt and instructing the LLM to follow the given schema.

The LLM had difficulties in identifying the tables and columns the user targeted in the question given the large number of tables with the same or similar column names (e.g. a 'testtype' or 'test_type' column is present in most of the sub-schema parent_tables of the DB). This often caused hallucinatory behavior from the LLM - the most severe one being to assign columns of a first table in a query to a second table (those columns not existing in the second table), creating an erroneous query which delivered an SQL error

In order to address this major issue we took two approaches:

1. Sub-schema selection: as the Vitic DB is structured on sub-schemas (a parent table complemented by child tables) we designed a sub-schema selector which generates a score by semantically comparing the embedded version of the question to the embedded version of `subschema_definitions,` boosted by keywords which identify each sub-schema. Once the sub-schema is selected (below we have the most referenced sub-schema: CARCINOGENICITY) the LLM gets to choose the table+columns combination for producing the query, drastically reducing the hallucinations in allocating the wrong column to the table or inventing columns outside the schema.
2. Schema / metadata clarity: explicitly state before each component of the database schema its label: before each column state "column_name:", before the table and column aliases state: table_alias and column_alias as it reduces hallucination, forcing the LLM to correctly identify the column+table relationship.



Compound_Information
meta_value
VITIC_LHASA.STRUCTURES
VITIC_LHASA.SYNONYMS

Parent	Child
meta_value	meta_code
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.CIT_CARCINOGENICITY
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.FT_CARCINOGENICITY
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.CARC_DOSE_RESP
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.CARC_OBSERVATIONS
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.CARC_RELIABILITY
VITIC_LHASA.CARCINOGENICITY	VITIC_LHASA.CARC_TUMOURS

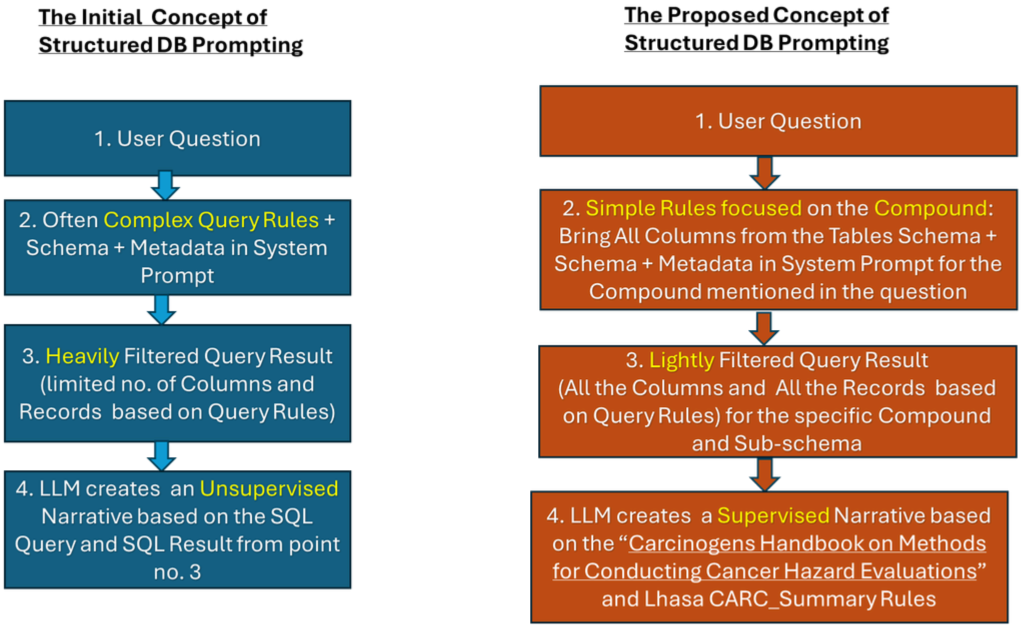
Generating accurate answers to natural language questions

A wide range of questions were asked against the LLM implementation. After getting rid of the most important sources of errors through the sub-schema selection and the schema refinement, an effort to modify the system prompts in order to answer the question in line with the two testers suggestions was carried out.

During the third sprint the scientist were presented with the LLM implementation. Their input changed our view of generating the LLM response as they had two major observations:

1. The response will need to be compound focused, stating in the title the analyzed compound common_name and the compound smiles code
2. The report will need to follow a reporting standard (Report on Carcinogens Handbook on Methods for Conducting Cancer Hazard Evaluations - indicated by Alex Cayley)

Based on the two observations we changed the prompting concept as presented bellow:


The Initial Concept of Structured DB Prompting
1. User Question
2. Often Complex Query Rules + Schema + Metadata in System Prompt
3. Heavily Filtered Query Result (limited no. of Columns and Records based on Query Rules)
4. LLM creates an Unsupervised Narrative based on the SQL Query and SQL Result from point no. 3
The Proposed Concept of Structured DB Prompting
1. User Question
2. Simple Rules focused on the Compound: Bring All Columns from the Tables Schema + Schema + Metadata in System Prompt for the Compound mentioned in the question
3. Lightly Filtered Query Result (All the Columns and All the Records based on Query Rules) for the specific Compound and Sub-schema
4. LLM creates a Supervised Narrative based on the "Carcinogens Handbook on Methods for Conducting Cancer Hazard Evaluations" and Lhasa CARC_Summary Rules

The major change between the two concepts comes from the fact that the SQL Query will have a minimum number of compulsory columns that are required by the Reporting standard. This is achieved by instructing the sub-schema system prompt which columns need to be included in the report. For example, in the `‘CARCINOGENICITY-prompt’` located here, columns like `'testtype', 'species', 'exposuretime', 'experimenttime', 'route', 'result', 'reliability', 'pvalue', 'goldtd50', 'observation2'` as `'tumor_type'` and `'site2'` as `'organ'` are indicated as compulsory.

Also all of the results per key columns are being included in the report, as the next step of the LLM process (the summation of the results) needs all the results per the key columns, as indicated in the system prompt:

```
“DO NOT INCLUDE the columns: 'testtype', 'route', 'result', 'reliability'
from the table vitic_lhasa.carcinogenicity in any 'AND' or 'WHERE' SQL Clause
when constructing the query.”
```

Prompt optimization

The next step in the Structured DB prompting proposes to instruct the LLM to summarize the results generated per Compound and sub-schema (e.g.: Benzo[a]pyrene + CARCENIGENICITY sub-schema) in a supervised manner as indicated in the `‘CARCINOGENICITY-narrative_prompt’` located here.

Hallucination gets eliminated for the per compound reports as all of the information for the specific compound is retrieved through the sql query, later the LLM generates a supervised report drawn from the query results guided by the: 'Guidelines' and the 'Study Findings' format, both indicating the scientific format of the report.

Vitic data quality and optimization recommendations @Orest Cramar

The internal structure of Vitic and the schematisation of some of its core data elements could be improved to make it easier to use in this and other analytic contexts (@Orest Cramar please elaborate).
Once we I have insulated the sub-schemas from the general schema most of the issues regarding the confusion regarding similar or same column names have been solved. By analyzing the Vitic sub-schemas if have observed the following:

1. **Reflections on Vitic Data Structure**

Strengths

- Normalized Schema: The Vitic schema is well-normalized. Core entities are separated into discrete tables (e.g., `carcinogenicity`, `carc_dose_resp`, `carc_observations`), reducing redundancy and improving maintainability.
- Clear Parent-Child Relationships: The consistent use of `parent_luid` as a foreign key to `carcinogenicity.luid` across multiple tables (e.g., `carc_dose_resp`, `carc_reliability`, etc.) facilitates easy traversal and aggregation of related toxicology study data.
- Rich Domain-Specific Tables: Specialized tables like `carc_tumours`, `carc_observations`, and `cit_carcinogenicity` indicate a domain-aware structure tailored to toxicological endpoints and scientific evidence collection.
- Linkage to Structures: The linkage from `carcinogenicity.structure_luid` to `structures.luid` and further to `synonyms` helps support chemical identification across naming systems, important for compound-based queries.

Limitations

- Potential Over-Normalization: The deep relational hierarchy may pose challenges for real-time query generation, particularly for models needing to construct multi-join SQL queries on the fly.
- Limited Encapsulation of Context: Tables capture measurements and metadata but often lack contextual fields (e.g., study purpose, outcome significance) that would be helpful for nuanced reasoning.
- No Explicit Linking Between Observation IDs Across Tables: Although many tables include `observationid`, it's not defined as a foreign key or clearly connected—possibly reducing relational clarity or increasing ambiguity.

2. Metadata Quality Assessment

Positives

- Standard Field Patterns: Almost all tables contain metadata fields like `vdb_created_date`, `vdb_modified_date`, `vdb_created_by`, and `vdb_modified_by`. This promotes traceability and supports audit and update tracking.
- Semantic Aliases: Many columns include aliases (e.g., `"Observation"`, `"Dose Unit"`, `"Klimisch Score"`), offering user-friendly labels that will help in prompt generation or UI design.
- Consistent Primary Keys: The use of `luid` as a universal primary key across all tables ensures internal consistency and simplifies record referencing.

Gaps

- Lack of Data Provenance or Source Context: Beyond citations (`cit_carcinogenicity`), there's little metadata explaining where data originated, under what conditions it was extracted, or with what assumptions.
- Sparse Description of Categorical Fields: Categorical fields like `"GLP"`, `"Test Type"`, or `"Lhasa Reliability Grade"` are not described with expected vocabularies or controlled terminology, which may hinder interpretability and automated parsing.
- Missing Unit Constraints: While fields like `doseunit` exist, there's no enforcement or metadata-driven catalog of units, potentially risking inconsistency across records.

3. Usefulness for LLM-Powered Prompting Model for Toxicology Scientists

High Utility Features

- Structured Representation of Carcinogenicity Data: The breakdown of a study's outcome into observations, tumors, dose-response data, and reliability allows a model to synthesize multi-dimensional answers.
- Well-Aligned with Domain Language: Field names, aliases, and categories reflect domain concepts (e.g., TD50, GLP, Lhasa Reliability), making it easier to construct semantically rich, human-readable responses.

- Linked Chemical Entity Representation: The ability to traverse from `structures` → `carcinogenicity` → various observations enables chemical-centric queries ("What is the carcinogenicity profile of Compound X?").
- Citation Integration: Availability of PMIDs and DOIs in `cit_carcinogenicity` allows the model to cite evidence or back up findings with literature references.

Challenges for LLM Integration

- Navigating a complex schema—such as generating precise SQL queries from natural language—demands multi-step reasoning, particularly when integrating elements like dose-response data with observations and reliability qualifiers.
- Potential Ambiguity in Observations: The presence of loosely defined fields (e.g., `observationid`, `site2`, `observation2`) without standardized taxonomies may introduce confusion during entity resolution or when matching across studies.
- Free Text Handling: The `ft_carcinogenicity` table contains unstructured data. While valuable, it requires separate NLP pipelines for extraction, summarization, or embedding into vector-based retrieval.
- Need for Ontology Alignment: For maximum interpretability and integration with external knowledge bases (e.g., MeSH, PubChem, OECD Guidelines), the schema would benefit from linking categorical values to ontologies.

Unstructured data

Data quality

Besides the large number of duplicate for the Cosmetic Ingredient Review (CIR) documents, the data was well curated.

Practical utility of this specific dataset

- CIR, SCCS, and PubMed data offer rich toxicological and regulatory insights, crucial for assessing compound safety.
- PubMed abstract only entries (the majority), while limited in detail, still contribute valuable signal-level insights for screening.
- The diversity of sources allows triangulation across regulatory-grade, peer-reviewed, and grey literature - essential in toxicology and pharmacology research workflows.
- As a large part of this literature is under copyright, exposing the information in a product might raise intelectual property rights concerns.
- Higher volumes (scaling from hundred thousands to millions of documents) would require redesign of the current storage option, for scalability and cost reasons.

Suggested optimisations

- Generate metadata for the CIR, CSSS, PubMed, Zotero papers.
- Standardize metadata across all categories of documents.
- Implement a document quality scoring (e.g. length, presence of dose/study details) to prioritize high-value documents.
- Semantic clustering - cluster for example Zotero and DOI papers by topic or compound using embedding techniques to accelerate search and synthesis
- Apply Graph RAG, to exploit the intrinsic graph structure of the metadata

Reflection on implication for using unstructured data more broadly

- The heterogeneity in the format and content depth (e.g. full text vs. abstract-only) poses integration challenges but mirrors real-world evidence variability.
- Unstructured data will further requires implementing robust curation pipelines in order to be truly actionable - especially in regulated domains where evidence traceability matters.
- The breadth of perspectives (academic, regulatory, exploratory) in unstructured data enriches analysis but will demand robust normalization and validation workflows.

Additional ideas for optimization

- Integration of citation networks to identify influential or corroborative studies.

- Optimization of retrieval by leveraging semantic chunking, different embeddings, and enhanced LLM instructions for the augmented generation.
- Leverage graph structure with multiple entity types from metadata (authors, articles, journals, publishers, themes, compounds) to enhance retrieval relevance and completeness (GraphRAG implementation).
- User feedback loops to flag misclassified or low-value content for continuous improvement.

Team utilisation and skillsets

The core team members on this project are detailed in this document: Team & Stakeholders

This project was executed with a combined team from Lhasa and Endava, with Lhasa providing core cloud engineering and DevOps capabilities, and Endava providing data science and generative AI expertise.

In a project of this nature, there are a number of intersecting but distinct activities which need to be executed:

Activity	Skills required
Provisioning data sources (Vitic and S3 files)	• DevOps
1. Building/deploying Knowledge Base	• Embeddings models & vector databases
2. Building and iterating the retrieval orchestration framework and Vitic querying	• Python development in Lambdas • GenAI multi-step reasoning • Agentic patterns • Relational database schemas & SQL
3. Optimising system prompts and other metadata	• Generative AI prompt design • SQL (for metadata)
4. Deploying versions of the framework for testing purposes	• DevOps
5. Running structured performance and UAT tests	• Testing • Model-based evaluation and related metrics

Items 3 and 4 on the list above represented the majority of "development" work in this project; but in contrast with a more traditional software development project, these tasks require a combination of technical, coding and AI skills. Furthermore, in the latter stage of the project, focus has shifted from orchestration design and iteration to prompt optimisation and testing, which has lessened the need for development skills. This has left Dave and Jamie with less to do in this phase, though they have achieved their learning goals of understanding how a generative AI system is put together. Additionally, Gabi and Orest have needed to refocus their energies on prompt optimisation - implementing further ideas on orchestration design has had to be put on hold to avoid unnecessary churn.

A key insight and learning from this project is that the resourcing of different parts of the project requires diverse skills, and it is not always possible to keep a fixed team with diverse capabilities busy. In thinking about how to resource projects like in this in future, Lhasa should think about the skillsets above and the points at which they become necessary for the tasks at hand at that stage in the project.

> For all of the task allocation and utilisation challenges that the team approach generated, a very important and valuable outcome of resourcing the project in the way we did is that it forced us to decompose the activities into definable stages and tasks and be somewhat explicit about the skills needed for those tasks. In a POC project like this, it can be common to have just one or two "core people" who perform many diverse tasks and achieve efficiency because they don't have to explain what they've done to other people, and have a deep understanding of the whole system. But this approach doesn't deliver very effective learning outcomes for the client: they might be left with a nice POC that does a useful thing, but they don't have a good enough understanding of how it was built, and more importantly, how to scale it out to achieve broader goals and how to deploy their own people to similar projects in the future.

Use case utility & applicability

The use case chosen for this PoC was always intended to to provide a meaningful challenge for an LLM-based AI system, as well as potentially delivering new capabilities in synthesizing structured and unstructured data that could be useful for either Lhasa scientists or scientists at Member companies as they research the safety of potential compounds to include in their products. The experience of implementing this kind of system has delivered some extremely useful insights.

- The system was most successful when asked about specific compounds, rather than broader questions about the kinds of tests conducted, or other overview questions. This largely reflected the nature of the underlying data - Vitic in particular is structured around specific compounds and designed to answer questions about them. Focusing on questions about specific compounds also limits the amount of data that is returned by the Vitic querying system, which keeps costs under control and helps with capacity constraints.
- The existence of Vitic's compound similarity search capability provides the opportunity to use a natural language interface to find and research compounds by similarity, but the application of this is limited to existing known compounds. This was not especially a focus of the POC work but may merit more investigation.
- The experience of making the Vitic querying system work, and the feedback from Alex Cayley, indicates that an useful capability may be to use generative AI purely as a summariser of data that has been retrieved from Vitic using a deterministic querying mechanism - for example, a "report builder" app (not using a natural language interface) which generates a structured, predictable query into Vitic, which is then summarised into a specific format using an LLM
- It is clear from testing that the unstructured corpus is adding value in prompt answering, but it has not been possible to determine the true utility of this additional data in providing extra context and richness to the answers coming from Vitic. A key learning in this area is that Vitic itself lacks any kind of durable connection to the research that went into its generation - there is only a very limited amount of study metadata in Vitic that can be used to flesh out its recommendations. It may be that a truly useful application of generative technology would be to generate this kind of link as part of the Vitic build process.
- Overall, relying on an open-ended natural language interface to both support general research and information exploration, and to generate format-specific outputs related to a particular compound, is likely too broad a use case to be effective We can think instead of two distinct use cases:
  - A natural-language (conversational) tool to aid research and information synthesis, perhaps fed by an existing corpus of information, or fed ad hoc by documents to be summarised at prompting time. This would be an aid to scientists looking to synthesise and summarise research more quickly and effectively
  - A structured UX tool that can be used to create a well-defined query into Vitic data and then summarise the resulting output with natural language, for example to create a textual report that can be sent to a regulator

Optimisation Approaches and Impact

Optimisation Approaches and Impact

Results of human evaluation

@Georgiana Madalina Vladeanu

Comparative feedback across models - Lhasa Gen AI - Production

Human v Test Runner feedback comparison - Lhasa Gen AI - Production

Retrieval Optimisation

Agentic RAG vs RAG Fusion

For unstructured data retrieval, two main approaches were used - RAG Fusion and Agentic RAG, as described earlier in this document.

Generation Optimisation

Model version

In addition to Claude Sonnet 3.0, a version of the system using Claude Sonnet 3.7 as its inference model was deployed and tested. The results were as follows:

[results - LGA-225]

System prompts @Orest Cramar

Prompt optimization involves striking a balance between too few instructions—leading to ambiguity—and too many instructions, which may include overlapping or conflicting rules that confuse the language model.

- Too few instructions → The LLM doesn't know what you're aiming for; responses may be too generic or off-target.
- Too many / conflicting instructions → The LLM might not know which instruction to prioritize, leading to inconsistent or incoherent output.
- The desirable middle ground is clear, concise guidance that defines the task without over-constraining the model.
- That is why I advice for using separate prompts (e.g. like dedicated narrative_prompts for “compound centered reports” vs “general question prompts” , as different objectives are being pursued by the the approaches) or dynamically modeled prompts (prompts stored in dedicated files). In both of these 2 approaches the reasoning of choosing which prompt to consider or how to dynamically optimize the prompt will be done based on business (scientific) rules derived from the interaction with the end users (scientists).
- The tool used to drive the prompt should be similar to the one described for Sub-schema selection: a sub-schema selector which generates a score by semantically comparing the embedded version of the question to the embedded version of `subschema_definitions,` boosted by keywords which identify each sub-schema.

Performance and Cost Optimisation

It became clear during this project that any production system would need to be optimised for performance and cost - for many queries, the response took 30 - 60 (or more) seconds to generate, and the cost of key system components such as OpenSearch and Bedrock could be substantial at higher query volumes. We didn’t have time to test the impact of optimisation approaches in these areas, but areas that would be important to investigate in a future project would be:

- Optimising the settings for OpenSearch, or considering an alternative vector database with a less expensive cost framework
- Exploring optimisation of the model calls at each stage of the query processing / retrieval orchestration flow - it may be that certain steps in the flow can be executed with less complex or expensive models, lowering the cost to service a particular question
- Exploring the relationship between the cost of serving answers and the utility of those answers and how this translates into decisions about the design of the retrieval orchestration flow. For example, an Agentic RAG approach which has more reflection loops might generate a marginally better answer, but this comes at considerable extra cost and time to respond.
- For a production system, it may be most cost-effective for higher query volumes to provision dedicated inference capacity.

Next Steps and Recommendations

Next steps and recommendations

Conclusions and next steps

We have decided not to continue to progress with this PoC past the end of June 2025, for several reasons:

- It has been difficult to generate useful answers across a sufficiently wide range of questions to make it attractive to progress with a member-facing conversational concept
- The POC has had a broad scope which, in the next phase of Lhasa’s AI journey, needs focusing
- It’s not clear what additional value the conversational interface has delivered on top of Vitic - Vitic already has a fairly comprehensive interface for querying the available information about a specific compound, including functionality such as the visual compound builder and similarity search which are hard to replicate in a natural language interface

- The value in enabling queries into Vitic that are qualitatively different from those which are enabled via its interface is not clear - for example, asking for all the compounds with studies over a certain length of time - and these kinds of query scenarios may be better served by making Vitic's data available as a more generic analytic dataset that could be accessed by an off-the-shelf tool such as Power BI.
- It's not been easy to establish the additional value (beyond that already summarised into Vitic calls) provided by the unstructured corpus of data, especially not in a nominal member-facing context. But we have established that it is possible to retrieve and present this data with an acceptably high degree of accuracy and precision.

Following internal discussions, Lhasa would like to progress with an internal effort to create a toolset to support the Lhasa Consult business, which utilises the information from a range of Lhasa products in order to provide expert opinion and regulatory filing support for Lhasa members. Taking this approach has a number of benefits:

- The AI-enabled toolchain will be only exposed to internal Lhasa employees, reducing the risk of inaccurate or hallucinatory answers making their way in front of Lhasa members
- Synthesizing data from multiple Lhasa products is currently time-consuming, presenting an obvious opportunity for automation that will improve Lhasa's margins on consulting work (or protect Lhasa's margins in the event that consulting work comes under price pressure in the future)
- This approach allows Lhasa to explore the best way to provision and use data from each of its products, likely using a protocol like MCP, and lends itself to growth in complexity and completeness over time
- The learnings about synthesising and summarising unstructured data will be valuable

Reusable components

It is unlikely that many of the components of the POC will be directly reused, but the following techniques that have been applied in the POC will be useful to reapply in future:

- Semantic indexing and retrieval optimisation for unstructured data (RAG Fusion and Agentic/reflective RAG)
- Metadata usage and SQL generation for unstructured data retrieval (the schema created for Vitic could prove very useful in future data management efforts)
- Use of Bedrock as an inference platform (model calls) and agentic platform (agent calls)

Further optimisation ideas

Although Lhasa is not planning to progress with this POC use case, the following optimisation ideas are useful to consider for other projects of this nature:

- **Unstructured content metadata:** Extracting and making use of metadata from unstructured documents (for example, document authors, institutions or methods) would improve the utility of this content when included in a generative system - for example by enabling the quality or authoritativeness of a document to be assessed. Lhasa should consider efforts to generate this kind of metadata for the content it uses to build its products.
- **Reference answer indexing:** Generative systems work better if they are provided with a rich set of reference answers (sometimes known as "golden" answers). A system that is designed to work across a broader range of question/answer types will benefit from access to a semantically-indexed store of many examples of "good" Q&A pairs, to include as few-shot learning examples in prompts.
- **Semantic chunking:** In this POC, the unstructured documents were chunked (broken up into pieces for indexing and retrieval) in a simple way, with each chunk a fixed size. Semantic chunking takes the structure and semantics of a source document into account, for example including a document's entire abstract as a single chunk, and then intelligently breaking the document up, so that retrieved chunks are more likely to be semantically consistent and distinct, improving retrieval effectiveness.
- **Dynamic prompting:** To meet a broader range of Q&A use cases, the system would benefit from the ability to dynamically alter the system prompts for key components based on an interpretation of the original question. For example, a question about a single compound would generate a different internal system prompt to one that asks to compare the qualities of many compounds, or many studies.
- **Multi-turn conversations:** The system as build is "single-turn", that is, the answer to a question is not included in the context (effectively, remembered) when the next question is asked. Adding the capability for "multi-turn" conversations (where conversation

history is remembered) could deliver additional value, but would require additional system design to prevent inefficiency and over-complication in the answers provided.

- **Alternative models:** It may be that other Foundation Models such as Llama, GPT or Gemini could provide better answers in some circumstances than Claude Sonnet; or that different models may work best together in a multi-agent system.