Development of a Retrieval-Augmented Generation Virtual Assistant for Enhanced Information Discovery at Rubin Observatory

Leanne P. Guy[a], Connor Yablonski[b], Aaron M. Meisner[c], Guillem Megias Homar[d], Merlin Fisher-Levine[e], Eman E. Ali[f], Tiger J. Hu[g], and Christopher W. Stubbs[h,i,j]

[a]NSF-DOE Vera C. Rubin Observatory / NSF NOIRLab, Casilla 603, La Serena, Chile
[b]University of Washington, Dept. of Astronomy, Box 351580, Seattle, WA 98195, USA
[c]NSF-DOE Vera C. Rubin Observatory / NSF NOIRLab, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[d]Division of Physics, Mathematics and Astronomy, California Institute of Technology, Pasadena, CA 91125, USA
[e]D4D CONSULTING LTD., Suite 1 Second Floor, Everdene House, Deansleigh Road, Bournemouth, UK BH7 7DU
[f]Swinburne University of Technology, Melbourne, Victoria, Australia
[g]Australian Astronomical Optics, Macquarie University, North Ryde, NSW, Australia
[h]Department of Astronomy, Center for Astrophysics, Harvard University, 60 Garden St., Cambridge, MA 02138, USA
[i]Center for Astrophysics, Harvard & Smithsonian, 60 Garden Street, Cambridge, MA 02138
[j]Department of Physics, Harvard University, 17 Oxford St., Cambridge MA 02138, USA

ABSTRACT

The NSF-DOE Vera C. Rubin Observatory will generate petabytes of data through the Legacy Survey of Space and Time (LSST) over the next decade, enabling discoveries across a broad range of astrophysical fields. Alongside these data products, Rubin maintains a large but heterogeneous collection of supporting documentation, including operational guides, technical notes, and scientific papers. Because this material is distributed across multiple platforms and formats, staff and scientists often struggle to efficiently locate accurate, up-to-date information. Many resources also reside on internal systems, limiting the ability of general-purpose language models to provide reliable answers to Rubin-specific questions. To address these challenges, we explore the use of Retrieval-Augmented Generation (RAG) to improve information discovery. We present a prototype RAG-based virtual assistant that delivers context-aware, factual, conversational access to Rubin's vast and heterogenous documentation ecosystem. The system integrates material from multiple sources and enables semantic search through a conversational interface, using Weaviate for embeddings, LangChain for query orchestration, and an OpenAI GPT model as the LLM backend. By grounding responses in domain-specific knowledge, the assistant reduces hallucinations, improves accuracy, and demonstrates the potential of RAG to enhance access to distributed knowledge, streamline workflows, and support effective use of LSST data products.

1. INTRODUCTION

The NSF-DOE Vera C. Rubin Observatory,[1] located at an elevation of 2647 m on Cerro Pachón in Chile, is a large-aperture, wide-field optical observatory whose prime mission is to conduct the 10-year Legacy Survey of Space and Time (LSST), beginning in 2026. The LSST will repeatedly image approximately 18,000 square degrees of sky every three nights using a 3.2-gigapixel camera mounted on the 8.4-meter Simonyi Survey Telescope. Each night, Rubin will generate 15–20 TB of raw data from which it will issue approximately 10 million transient alerts. Over its lifetime, the LSST is expected to catalog approximately 20 billion galaxies and 17 billion stars, producing a final catalog of approximately 15 PB and roughly 500 PB of image products across several major data releases, each accompanied by updated documentation, data product definitions, and software.

This scientific output is accompanied by an extensive body of supporting documentation, including technical notes, design documents, operational guides, data product definitions, software documentation, and community-facing tutorials, distributed across a wide variety of platforms: Confluence wikis, Jira tickets, GitHub repositories, the Rubin Community Forum*, Slack discussions, DocuShare archives, and numerous PDF reports. This fragmentation creates a significant challenge for both Rubin Observatory staff and the broader science community when attempting to locate accurate, up-to-date information. The challenge is compounded by the scale of the Rubin science community itself: approximately 10,000 scientists across 28 countries hold Rubin data rights, a regime in which traditional helpdesk-based user support models do not scale.[2] Rubin's approach to community science instead focuses on fostering a vibrant community supported by infrastructure such as the Discourse-based Rubin Community Forum, to crowdsource solutions and build a deep reservoir of shared expertise.[3]

Large language models (LLMs) offer a promising approach to this information-discovery challenge. Their ability to understand and generate natural language makes them well suited to serving as an intelligent interface to large, heterogeneous documentation ecosystems, allowing users to find information through conversation without needing to know where each piece of information resides. However, LLMs face fundamental limitations when applied to domain-specific questions about Rubin Observatory. LLMs compress vast amounts of knowledge during training into their weights, but this encoded knowledge is *static*: it cannot be updated as new information becomes available, and models can already be outdated by the time they are released. Because LLMs are trained on broad corpora, they often lack the *domain-specific* knowledge needed to answer questions about Rubin's data products, the Rubin Science Platform, or the LSST science pipelines,[4] and may generate plausible but factually incorrect responses, a phenomenon known as *hallucination*. Furthermore, much of the information critical to Rubin operations and science resides on internal or proprietary systems, making it entirely inaccessible to general-purpose models regardless of their training.

Retrieval-Augmented Generation (RAG) was introduced by Lewis et al.[5] to address precisely these limitations. Their key insight was that LLMs possess a form of implicit, *parametric* memory, that is, knowledge encoded in the model's weights during training, but that this memory is fixed and cannot account for information absent from the training corpus. By combining this parametric memory with an explicit, *non-parametric* memory in the form of a searchable document index, and fetching relevant passages from that index at query time, RAG enables an LLM to generate responses grounded in specific, up-to-date context rather than relying solely on its compressed internal knowledge. This hybrid approach was shown to significantly improve factual accuracy on knowledge-intensive tasks compared to a standalone LLM.[5] RAG was selected over model fine-tuning because the core challenge is not how the model responds but what information it can access, and because the knowledge base must evolve across data releases without expensive retraining.[6]

Building on this framework, we are developing a RAG-based virtual assistant designed to ground LLM responses in Rubin Observatory's knowledge bases, providing more accurate, current, and contextually relevant answers while reducing hallucinations. The goals are two-fold: to support Rubin project staff in rapidly and accurately retrieving information across the project's myriad platforms and formats; and to support the global Rubin science community in exploiting Rubin data products and services. This paper describes the motivation, architecture, and implementation of a prototype RAG virtual assistant for Rubin Observatory, the lessons learned during its development, and future directions.

2. MOTIVATION AND USE CASES

2.1 Supporting daily Rubin operations

Rubin project staff rely daily on a broad set of private collaboration resources, including Confluence wiki pages, Jira tickets, Slack discussions, Community Forum threads, proprietary documents, engineering drawings, and a long history of design documents in the internal DocuShare archive. A RAG-grounded virtual assistant that makes this distributed and rapidly evolving body of knowledge searchable through a single conversational interface would significantly reduce the time staff spend locating information and help prevent operational decisions from being made on the basis of outdated or superseded documents. Making this distributed and rapidly evolving body of knowledge searchable through a single conversational interface would significantly reduce the time staff spend

*https://community.lsst.org

locating information and help prevent operational decisions being made on the basis of outdated or superseded documents. A particular strength of RAG in this context is its ability to synthesize information across platforms. For example, a question such as *"what observing strategy was adopted for the LSSTCam science validation surveys?"* might require combining the technote reporting on the LSSTCam observing campaign strategy, Jira tickets recording the actual commands executed, and Slack threads where details were discussed. No single platform contains the complete answer; a RAG system searching across all three can assemble it. Similarly, ingesting the Simonyi Survey Telescope troubleshooting guide from Confluence would allow night-time observers to query telescope faults in natural language during time-critical operations. Key reference documents such as the LSST Science Book and the final telescope design document can likewise be made conversationally accessible, with answers traceable to their source. The virtual assistant would also support onboarding of new project staff, who must rapidly absorb institutional knowledge scattered across over 20 years of archived discussions, technical notes, and design decisions.

2.2 Supporting the science community

The Rubin RAG virtual assistant is also designed to support the global science community in exploiting Rubin data products. The Rubin Early Science Program[7] will release science-grade commissioning and early pre-survey data to support community preparations for LSST operations and enable high-impact science as early as possible. The first full LSST data release, DR1, is expected approximately two years after the start of survey operations. In the intervening period, a substantial volume of data will be made available through transient alerts, nightly processed images, and catalogs of single-epoch detections, as well as through the Data Preview program (DP1,[8] DP2) in the period leading up to DR1. Each of these releases represents an opportunity for early scientific discovery with Rubin data. During this same period, the community will also be familiarizing itself with Rubin's data products, services, access tools, and the algorithms used to produce the data products. A virtual assistant grounded in authoritative, up-to-date Rubin documentation can lower the barrier to entry, help users navigate unfamiliar systems, and accelerate the path from data access to scientific results.

The eventual goal is for the Rubin virtual assistant to answer technical questions and provide at least beginner-level coding assistance for working with the LSST Science Pipelines. A key challenge for users will be navigating the Rubin data model,[9] which comprises hundreds of columns across multiple catalog tables with Rubin-specific naming conventions, and constructing correct and efficient ADQL/TAP queries. For example, a user asking *"how do I select clean galaxy samples from the Object table?"* needs to know which flag columns to filter on to exclude objects with unreliable shape or light profile modeling, and how to apply signal-to-noise cuts to retain only well-detected galaxies. These column names are specific to Rubin and may change between data releases; a general-purpose LLM may hallucinate plausible but incorrect names, while a RAG system grounded in the current schema helps mitigate this risk. Similarly, a question such as *"how do I retrieve the coadded images for the ECDFS field from DP1?"* requires knowing the correct Butler[10] collection name, the filter designations available in DP1, and the current Butler API syntax, all details that may change between releases and are absent from general LLM training data. A RAG system grounded in the schema documentation, data product definitions, and worked examples from tutorials and Community Forum posts can provide reliable, validated guidance that a general-purpose LLM cannot. The Community Forum is a particularly valuable knowledge source: a user encountering an error running a tutorial notebook may find that the identical problem was already resolved on the Forum; the virtual assistant can surface that answer directly rather than requiring the user to search the Forum manually.

2.3 Documentation validation

A RAG virtual assistant also serves as a tool for validating Rubin's user-facing documentation. By creating a standard suite of prompts and systematically querying the virtual assistant, the team can identify gaps, inconsistencies, or ambiguities in existing documentation. If the virtual assistant cannot answer a question that should be answerable, this signals a documentation gap. If it produces an incorrect answer, this may indicate ambiguous or contradictory source material. For example, testing whether the virtual assistant correctly reports which filters are available in DP1, a subset of the full *ugrizy* set, can reveal whether the documentation clearly distinguishes DP1-specific content from general LSST descriptions. The approach can also detect whether tutorial notebooks reference deprecated API calls that no longer work against the current software stack, or whether

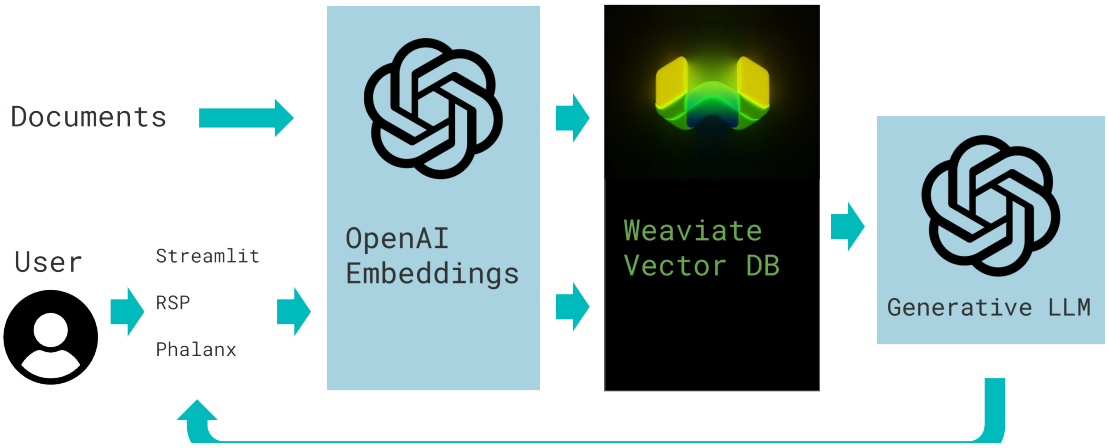

Figure 1. Architecture of the Rubin RAG virtual assistant showing the flow from document ingestion through embedding, vector storage, retrieval, and response generation. Documents from the Rubin knowledge base are loaded, chunked, and embedded using OpenAI's `text-embedding-3-large` model. Embeddings and text chunks are stored in the Weaviate vector database. At query time, a user query submitted via the Streamlit frontend is embedded using the same model; the top-k most similar chunks are retrieved and injected into the LLM prompt to produce a grounded response. The system is deployed on the Rubin Science Platform through the Phalanx infrastructure. (FIGURE WILL BE IMPROVED BEFOR SUBMISSION

documented Butler collection names match the collections actually deployed on the RSP. Rubin's Community Science Team has already explored using Vale for automated style verification of user-facing documentation; the virtual assistant provides a complementary, content-level validation approach.

3. ARCHITECTURE

We adopt the modular variant of the RAG architecture,[11] in which the retrieval, augmentation, and generation components are independently built, tested, and swappable. This modularity has been essential during development, allowing us to experiment with different embedding models, chunking strategies, and LLM backends without redesigning the full system. Figure 1 shows the overall system architecture.

3.1 Overview: the RAG framework

The three words in Retrieval-Augmented Generation describe the three stages of the system:

- **Retrieval.** Given a user query, the system searches an external knowledge base to find the most relevant document fragments. The user's query is converted into a vector embedding and compared against stored document embeddings using cosine similarity to identify the top-k most semantically relevant chunks.
- **Augmentation.** The retrieved chunks are assembled into a structured context block and combined with the original user query to form an augmented prompt. A system prompt instructs the LLM to base its answer on the provided context and to indicate when the available documentation does not contain sufficient information, rather than generating a speculative response. This augmentation step is what distinguishes RAG from a standard LLM interaction: the model receives domain-specific context alongside the question.
- **Generation.** The augmented prompt is passed to a generative LLM, which produces a natural language response conditioned on both the user's question and the retrieved context. Because the response is grounded in specific, retrieved documents rather than the model's compressed parametric knowledge alone, the output is more factually accurate and traceable to its sources.

The following subsections describe the specific components selected for each stage, together with the frontend, deployment infrastructure, and platform integration that connect them into a working service.

3.2 Retrieval: Weaviate vector database

The retrieval stage depends on an efficient, searchable store of document embeddings. We use Weaviate as the vector database. Weaviate organizes data in "collections" that store text chunks, associated metadata (source URL, document identifier, page number, creation date), and their vector embeddings together. This co-location of vectors, text, and metadata allows the retrieval component to efficiently compare the vector embedding of a user query against the stored embeddings using cosine similarity, and to return both the matching text and its provenance. Weaviate was selected for its native integration with LangChain, its support for multiple vectorization backends, and its suitability for deployment on Kubernetes.

3.3 Retrieval and generation: embedding and generative models

The system uses two separate OpenAI models serving distinct roles in the retrieval and generation stages. The `text-embedding-3-large` model is used as the *vectorizer* in the retrieval stage, converting both document chunks and user queries into high-dimensional vector representations that capture semantic meaning. The same embedding model is used at ingestion time and at query time, ensuring that documents and queries occupy the same vector space and that similarity comparisons are meaningful. The `gpt-3.5-turbo` model serves as the *generative LLM* in the generation stage, producing natural language responses based on the augmented prompt containing the user's query and the retrieved context. Both models are accessed via the OpenAI API through keys provided to the Weaviate client and LangChain.

While we use OpenAI for both components during development, the modular architecture allows alternative embedding models and LLMs to be substituted. We plan to benchmark alternative LLMs including more recent OpenAI models, Anthropic's Claude, and open-source alternatives to optimize the trade-off between response quality, latency, and cost.

3.4 Augmentation and orchestration: LangChain

LangChain is an open-source Python library that provides the integration layer connecting all three RAG stages. During ingestion, LangChain manages the document loading, chunking, and embedding pipeline that populates the Weaviate vector store (Section 4). At query time, LangChain orchestrates the full RAG cycle: embedding the user query, retrieving relevant chunks from Weaviate, constructing the augmented prompt with retrieved context, and routing the augmented prompt to the generative LLM. Its modular design allows individual components, including loaders, splitters, embedding models, vector stores, and LLMs, to be swapped independently, which has been essential for the iterative experimentation and optimization described in Section 7.

3.5 Frontend: Streamlit

The user-facing interface is implemented as a Streamlit application providing a conversational chat interface. The interface includes selectable source filters, e.g Confluence, Jira, LSST forum, that allow users to control which document collections are searched for a given query. The application is lightweight, requiring relatively little code while providing an intuitive chat paradigm. Figure 2 shows the interface as deployed on the RSP.

3.6 Deployment infrastructure

The virtual assistant is deployed using Rubin Observatory's standard deployment model, Phalanx.[12] Phalanx is built on a layered infrastructure stack: Kubernetes for container orchestration, Argo CD for GitOps-based continuous delivery, Helm for packaging Kubernetes applications, and the Safir framework for Rubin-specific service development. The virtual assistant is containerized via a Dockerfile in the repository and deployed through the same GitOps workflows used for all other RSP services.

The Phalanx-based deployment model means that the virtual assistant's configuration is managed as code in the Phalanx repository, enabling reproducible deployments, version-controlled updates, and rollback capability.

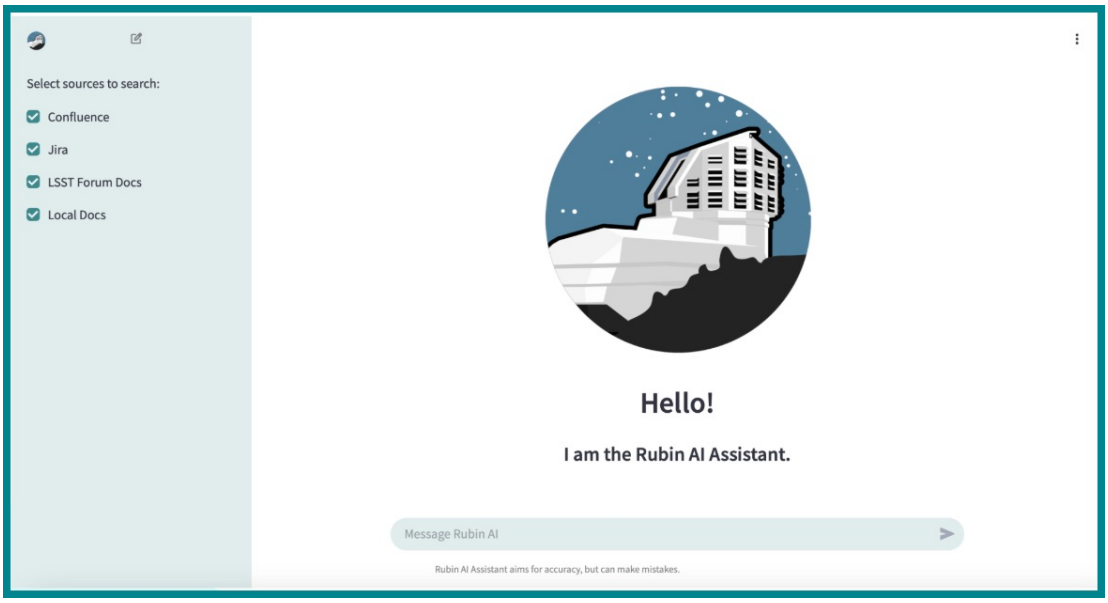

Figure 2. The Rubin AI Assistant conversational interface, implemented in Streamlit and deployed on the Rubin Science Platform. The left sidebar shows the source-filter panel (Confluence, Jira, LSST Forum Docs, Local Docs), allowing users to target specific document collections. Users enter natural-language questions in the message bar; the system retrieves relevant document chunks and returns a grounded response with source citations.

3.7 Integration with the Rubin Science Platform

From the outset, the RAG virtual assistant was designed to be deployed as a service within the Rubin Science Platform (RSP), following the same model used for all other RSP services, at both the user-facing cloud RSP[†] and the on-premises United States Data Facility (USDF) at SLAC National Accelerator Laboratory.[13]

The LSST will accumulate approximately 50 PB of data products by the end of year 1, growing to roughly 500 PB by year 10; far too large for users to download and analyse locally. The RSP is Rubin's answer to this challenge, providing cloud-based access to LSST data products and services through three interfaces: the Portal (browser-based data discovery), Notebooks (Jupyter-based analysis), and Web APIs (programmatic access via Virtual Observatory interfaces). Deploying the virtual assistant within this same infrastructure provides several important advantages:

- **Co-location with the data**: Users interact with the virtual assistant in the same environment where they access and analyze LSST data products, enabling a tightly integrated workflow in which documentation assistance and data analysis are available side by side.
- **Shared authentication and access control**: The virtual assistant inherits the RSP's identity management and access control infrastructure, ensuring that private or restricted resources are accessible only to authorized users without requiring separate login.
- **Operational monitoring**: The virtual assistant benefits from Rubin's existing service monitoring and alerting infrastructure, reducing the operational burden of running an additional service.
- **Dual-environment coverage**: Deployment at both the cloud RSP and the USDF ensures that the virtual assistant is accessible to users regardless of which environment they use for data analysis.

[†] https://data.lsst.cloud

3.8 Codebase

The codebase is developed in Python and maintained as open-source software in the `lsst-dm/rubin_rag` GitHub repository. The repository contains the Streamlit application, the Weaviate connection and query layer, the data ingestion code, and Jupyter notebooks used for prototyping and testing. The deployment configuration is managed through a corresponding branch in the Phalanx repository. The project is released under the MIT license.

4. DATA INGESTION PIPELINE

The data ingestion pipeline transforms raw documents from diverse sources into vector embeddings stored in Weaviate, ready for retrieval. The pipeline consists of three main steps: loading and cleaning, chunking, and embedding.

4.1 Data Sources

The Rubin RAG system ingests documents from the following sources:

- **Rubin technical documents**: Design documents, studies, reports, requirements specifications, and technical notes authored in a variety of formats including LaTeX, reStructuredText, Markdown, and occasionally Word. Where the original source is available it is ingested directly; a rendered PDF is used only when no source format can be obtained.
- **GitHub repositories**: Including the LSST codebase and its documentation, Data Release tutorials and documentation;
- **Jira tickets:** Technical project management and issue-tracking records containing design discussions, implementation decisions, and bug reports. Administrative and access-restricted tickets are excluded.
- **Rubin Community Forum**: Discourse-based forum threads, including resolved support questions that represent practical knowledge and experience;
- **Confluence:** Selected internal wiki pages containing meeting notes, design discussions, and operational procedures. Access-restricted and private spaces are excluded from ingestion.
- **Slack channels:** Selected technical discussion channels containing decisions, troubleshooting conversations, and domain expertise relevant to operations and data processing. Private and administrative channels are excluded; and
- **DocuShare:** Legacy document management archive containing historical design documents, engineering drawings, requirements documents, and early project records, often in non-standard formats. Ingesting this material presents particular challenges due to format diversity but provides access to institutional knowledge not available elsewhere.

4.2 Loading and Cleaning

We use LangChain document loaders to extract text from files or to parse data returned by APIs. For PDFs, we use the `PyMuPDFLoader`, which parses text and tables effectively but does not recover non-text information embedded in images. Each loader outputs a list of LangChain `Document` objects containing the extracted text and associated metadata (source file, creation date, page number). For GitHub repositories we use the `GithubFileLoader`, which retrieves files directly from a specified repository and branch via the GitHub API; for Jira we query the Rubin Jira REST API directly to retrieve ticket content and metadata. In both cases the result is a list of LangChain `Document` objects containing extracted text and metadata, ready for downstream chunking and embedding.

Several areas for improvement have been identified. The current pipeline loses information encoded in figures and diagrams, which are prevalent in Rubin documentation; vision-language models could generate text descriptions of these for embedding, making visual content searchable. For scanned documents and image-based

PDFs, particularly prevalent in the DocuShare legacy archive where non-PDF source formats are often unavailable, we are evaluating OCR and multimodal extraction techniques to recover text that standard PDF parsers cannot access. Where source formats are available, we ingest them directly: LaTeX technotes are parsed from their LaTeX source rather than from rendered PDFs, and Markdown and reStructuredText documents handle tabular content reliably without the extraction issues that affect PDF tables. For conversational sources such as the Community Forum, Jira, and Slack, raw ingestion produces noisy content mixing resolved answers with tangential discussion; extracting clean, structured records, such as resolved question-answer pairs from forum threads or decision records from Jira tickets, would substantially improve retrieval quality. For Jira in particular, distinguishing currently relevant tickets from obsolete ones is an open challenge; approaches under consideration include filtering by ticket status and recency, weighting by activity level, and applying time-decay factors during retrieval so that older, unresolved tickets are ranked lower than recent, resolved ones. Finally, as the corpus grows, deduplication across sources becomes important: the same information often appears in a Jira ticket, a Confluence page, and a technote, and retrieving contradictory versions of the same content degrades response quality

4.3 Chunking

Loaded documents are split into chunks using LangChain's `RecursiveCharacterTextSplitter`, which divides text by character count while preferring to split at paragraph breaks, then line breaks, then sentence boundaries. The current prototype uses a chunk size of 1000 characters with an overlap of 50 characters between adjacent chunks.

Chunk size is a critical trade-off:

- Chunks that are *too small* may lack sufficient context for the LLM to generate a meaningful answer, even when the correct passage is retrieved.
- Chunks that are *too large* cause the query embedding and chunk embedding to diverge semantically, reducing retrieval precision and risking retrieval from the wrong part of the document.

We plan to evaluate several alternatives to the current character-based chunking strategy, including "small-to-big" retrieval,[14,15] which uses smaller chunks for precise retrieval while providing larger parent chunks to the LLM for generation. This decouples the granularity used for retrieval from the context provided to the generator, addressing both failure modes simultaneously. We also plan to evaluate semantic chunking,[16] which detects topic boundaries by measuring embedding similarity between adjacent sentences rather than splitting at fixed character counts, an approach that has been shown to improve retrieval recall over fixed-size methods.[17] Document-structure-aware and code-aware chunking, best practices now implemented in LangChain's specialized splitters,[18] use the inherent structure of source material to define chunk boundaries: section headers in technotes, function definitions in API documentation, and cell boundaries in Jupyter notebooks. Cell-level chunking of notebooks is particularly important, as tutorial notebooks are among the most valuable sources for community users and each cell typically represents a self-contained, executable step in a scientific workflow that naive character-based splitting would break into unusable fragments.

4.4 Embedding

After chunking, documents are embedded using the OpenAI `text-embedding-3-large` model via LangChain's `WeaviateVectorStore`, which creates and populates Weaviate collections. Each collection stores the chunk text, metadata, and vector embedding together, enabling efficient similarity search at query time. The same embedding model is used for both documents at ingestion time and for user queries at retrieval time, ensuring that both are represented in the same semantic vector space.

Several directions for improving embedding quality are under consideration. The current pure vector similarity search can miss exact term matches when users query specific column names, error messages; we plan to evaluate hybrid retrieval combining dense (vector) and sparse (BM25 keyword) search, a capability Weaviate supports natively. We also plan to benchmark alternative embedding models, since the general-purpose `text-embedding-3-large` may not optimally represent Rubin-specific terminology where common words carry

domain-specific meanings (e.g., "Butler" as a data access framework, "tract" and "patch" as sky tiling units). Finally, we are exploring metadata-enriched embedding, in which document context such as the title, section header, and source type is prepended to each chunk before embedding, so that the vector representation captures provenance alongside content.

5. QUERY PROCESSING AND RESPONSE GENERATION

When a user submits a query through the Streamlit interface, the system processes it through the following steps:

1. **Query embedding**: The user's query is embedded using the same `text-embedding-3-large` model used during ingestion, producing a query vector in the same semantic space as the stored document embeddings.
2. **Retrieval**: The query vector is compared against all stored chunk embeddings in Weaviate using cosine similarity search. The top-k most similar chunks are retrieved, along with their source metadata (document title, URL, page number). Users can filter which source collections are searched, allowing targeted retrieval when the relevant domain is known.
3. **Prompt augmentation**: The retrieved chunks are injected into the LLM's prompt alongside the original user query, forming an augmented prompt that grounds the LLM's response in specific retrieved context.
4. **Response generation**: The `gpt-3.5-turbo` model generates a natural language response conditioned on both the user's query and the retrieved context. The response is streamed back to the user through the Streamlit interface.
5. **Source citation**: The interface displays the source documents from which context was retrieved, enabling the user to verify the information and consult the primary sources directly.

6. EVALUATION AND CURRENT STATUS

Evaluating RAG systems is non-trivial: standard LLM benchmarks do not measure retrieval quality, and the correctness of a generated response depends on both the relevance of the retrieved chunks and the quality of generation conditioned on that context.

6.1 Validation query set

We have developed a validation query set categorized into three types:

1. Queries answerable from general LLM knowledge without any domain-specific retrieval, to verify baseline behavior, for example: *"What is a color-magnitude diagram?"* or *"What is the difference between AB and Vega magnitude systems?"*;
2. Queries that require Rubin-specific documentation to answer correctly, to verify that retrieval is effective and the system outperforms the base LLM, for example: *"What flags should I use to select clean galaxy samples from the DP1 Object table?"* or *"What is the Butler in the LSST Science Pipelines?"*; and
3. Queries that probe known failure modes, such as questions about very recent events, highly specific technical details, or information that exists only in private systems, for example: *"What is the magnitudeError column in the DP1 Object table?"* (a non-existent column) or *"Can I use the DP0.2 tutorials to work with DP1 data?"* (version confusion).

For each query, we define either an expected answer or an expected behavior (e.g., declining to answer or requesting clarification). Together, the three types assess whether the system handles general knowledge correctly, whether retrieval meaningfully improves responses on Rubin-specific questions, and whether the system fails gracefully when it lacks the information to answer. Running each query with and without retrieval enabled quantifies the improvement contributed by the RAG component.

6.2 RAGAS Framework

For automated evaluation, we use the RAGAS framework,[19] an open-source toolkit that provides reference-free metrics specifically designed for RAG systems. RAGAS evaluates system quality along four axes: *faithfulness*, *answer relevancy*, *context precision*, and *context recall*. Together these cover both the retrieval and generation stages of the pipeline, allowing failures to be attributed to the appropriate stage. RAGAS also supports the definition of custom metrics, enabling domain-specific evaluation criteria to be added alongside the built-in ones. For Rubin, this opens the possibility of metrics that verify whether a response correctly identifies the relevant data product, uses valid catalog column names, or produces syntactically correct ADQL queries—aspects of correctness that general-purpose metrics cannot assess.

6.3 Current performance and diagnosis

Initial end-to-end evaluation reveals that response quality does not yet meet the standard required for community deployment. While the core pipeline is functional, applying the RAGAS metrics to the validation query set shows that degradation arises from multiple interacting sources across the full pipeline rather than any single bottleneck.

On the *retrieval* side, the current fixed-size chunking strategy produces chunks that frequently split relevant context across boundaries, reducing context recall. The small-to-big retrieval approach and semantic chunking strategies described in Section 4.3 are being evaluated as alternatives. Embedding model selection also plays a role: the general-purpose `text-embedding-3-large` model may not optimally represent Rubin's highly technical vocabulary, including observatory-specific acronyms, instrument names, and catalog schema terminology.

On the *generation* side, systematic prompt engineering has not yet been performed. The current system prompt and retrieval-augmented prompt template were constructed heuristically; structured optimization is planned. We are also evaluating alternative LLM backends beyond `gpt-3.5-turbo`, as more recent models offer stronger reasoning over retrieved context, better code generation capabilities, and more reliable adherence to system prompt instructions.

The composition and coverage of the ingested corpus itself is also under review. Gaps in source coverage, inconsistent document quality, and stale content all reduce the system's ability to ground responses in authoritative information. Duplication of content across sources can saturate the retrieval results with redundant chunks, displacing more relevant content from the top-k results and, when different versions are out of sync, introducing contradictory context that degrades response quality.

7. FUTURE DIRECTIONS

The current prototype demonstrates the viability of the RAG approach for Rubin Observatory and has established the core infrastructure and deployment tooling. The following sections describe planned near-term improvements and the longer-term vision for the system.

7.1 Near-term improvements

- **Chunking strategy optimization.** The current fixed-size chunking is a primary source of retrieval degradation, producing chunks that split relevant context across boundaries and reduce recall. Improved chunking strategies (Section 4.3) are expected to increase retrieval precision, ensure that retrieved chunks contain complete and coherent answers, and preserve executable code blocks intact for the coding assistance use case.

- **Prompt engineering.** Systematic optimization of the system prompt and retrieval-augmented prompt template to steer the model toward more accurate, well-structured, and appropriately scoped responses. The system prompt governs how the LLM interprets retrieved context, when it declines to answer, and how it attributes its sources; careful tuning of these instructions has a significant impact on response quality.

- **Model benchmarking.** Evaluating alternative LLM and embedding model backends, including more recent OpenAI models, Anthropic's Claude, and open-source alternatives, to optimize the trade-off between response quality, latency, and cost.

- **Corpus expansion and automation.** Expanding the ingested corpus to cover all documentation relevant to each data release and automating the ingestion pipeline with update triggers linked to source platforms (GitHub webhooks, API polling for Jira and Confluence, Slack channel ingestion).
- **Productizing.** Transitioning from prototype to a production-quality service with robust monitoring, logging, usage analytics, and automatic re-ingestion of updated and new data sources.

7.2 Towards agentic RAG

The current prototype follows a fixed pipeline: each query passes through retrieval, augmentation, and generation in a single pass without the ability to iterate, reformulate queries, or decompose complex multi-step questions. This limits effectiveness on queries that require reasoning across multiple sources or iterative refinement of a response.

Agentic RAG represents a natural evolution by introducing an explicit control layer that coordinates how the system reasons over external context. Rather than following a fixed pipeline, an agentic system can dynamically decide which collections to search, validate retrieved information for consistency, reformulate queries when initial retrieval is insufficient, invoke external tools, and iteratively refine responses.

These capabilities are well suited to the most ambitious Rubin use cases described in Section 2: schema-aware query construction that verifies column names against the current catalog schema; tutorial-guided science workflows that assemble multi-step code examples from different sources; and cross-platform synthesis that combines information from technotes, Jira tickets, and forum posts into a coherent answer. We plan to investigate agentic RAG architectures as the system matures beyond the current prototype.

8. SUMMARY

We have presented the design and implementation of a RAG-based virtual assistant for information discovery at the Vera C. Rubin Observatory, deployed as a service on the Rubin Science Platform. The system addresses a real operational need: enabling both Rubin Observatory staff and the Rubin science community to efficiently locate accurate information across a large, fragmented, and continuously evolving documentation ecosystem. The prototype is functional and deployed on the RSP at both the cloud-based US Data Access Center and the on-premise USDF at SLAC, and is being prepared to support the community during the Early Science period. As the LSST survey progresses and both data volume and documentation grow across successive releases, the importance of this capability will only increase. Rather than a static documentation virtual assistant, a value proposition that has eroded as general-purpose LLMs have become web-connected, the longer-term vision is a science enablement assistant that helps users go from a science question to working code against real Rubin data, grounded in authoritative and current sources. The RAG value is not in knowing what the documentation says; it is in knowing which specific API call, column name, or query pattern will actually work on the current data release.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreements AST-1258333 and AST-2241526 and Cooperative Support Agreements AST-1202910 and AST-2211468 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory managed by Stanford University. Additional Rubin Observatory funding comes from private donations, grants to universities, and in-kind support from LSST-DA Institutional Members.

REFERENCES

[1] Ivezić, Ž., Kahn, S. M., Tyson, J. A., Abel, B., Acosta, E., Allsman, R., Alonso, D., AlSayyad, Y., Anderson, S. F., Andrew, J., Angel, J. R. P., Angeli, G. Z., Ansari, R., Antilogus, P., Araujo, C., Armstrong, R., Arndt, K. T., Astier, P., Aubourg, É., Auza, N., Axelrod, T. S., Bard, D. J., Barr, J. D., Barrau, A., Bartlett, J. G., Bauer, A. E., Bauman, B. J., Baumont, S., Bechtol, E., Bechtol, K., Becker, A. C., Becla, J., Beldica, C., Bellavia, S., Bianco, F. B., Biswas, R., Blanc, G., Blazek, J., Blandford, R. D., Bloom, J. S., Bogart, J., Bond, T. W., Booth, M. T., Borgland, A. W., Borne, K., Bosch, J. F., Boutigny, D., Brackett, C. A., Bradshaw, A., Brandt, W. N., Brown, M. E., Bullock, J. S., Burchat, P., Burke, D. L., Cagnoli, G., Calabrese, D., Callahan, S., Callen, A. L., Carlin, J. L., Carlson, E. L., Chandrasekharan, S., Charles-Emerson, G., Chesley, S., Cheu, E. C., Chiang, H.-F., Chiang, J., Chirino, C., Chow, D., Ciardi, D. R., Claver, C. F., Cohen-Tanugi, J., Cockrum, J. J., Coles, R., Connolly, A. J., Cook, K. H., Cooray, A., Covey, K. R., Cribbs, C., Cui, W., Cutri, R., Daly, P. N., Daniel, S. F., Daruich, F., Daubard, G., Daues, G., Dawson, W., Delgado, F., Dellapenna, A., de Peyster, R., de Val-Borro, M., Digel, S. W., Doherty, P., Dubois, R., Dubois-Felsmann, G. P., Durech, J., Economou, F., Eifler, T., Eracleous, M., Emmons, B. L., Fausti Neto, A., Ferguson, H., Figueroa, E., Fisher-Levine, M., Focke, W., Foss, M. D., Frank, J., Freemon, M. D., Gangler, E., Gawiser, E., Geary, J. C., Gee, P., Geha, M., Gessner, C. J. B., Gibson, R. R., Gilmore, D. K., Glanzman, T., Glick, W., Goldina, T., Goldstein, D. A., Goodenow, I., Graham, M. L., Gressler, W. J., Gris, P., Guy, L. P., Guyonnet, A., Haller, G., Harris, R., Hascall, P. A., Haupt, J., Hernandez, F., Herrmann, S., Hileman, E., Hoblitt, J., Hodgson, J. A., Hogan, C., Howard, J. D., Huang, D., Huffer, M. E., Ingraham, P., Innes, W. R., Jacoby, S. H., Jain, B., Jammes, F., Jee, M. J., Jenness, T., Jernigan, G., Jevremović, D., Johns, K., Johnson, A. S., Johnson, M. W. G., Jones, R. L., Juramy-Gilles, C., Jurić, M., Kalirai, J. S., Kallivayalil, N. J., Kalmbach, B., Kantor, J. P., Karst, P., Kasliwal, M. M., Kelly, H., Kessler, R., Kinnison, V., Kirkby, D., Knox, L., Kotov, I. V., Krabbendam, V. L., Krughoff, K. S., Kubánek, P., Kuczewski, J., Kulkarni, S., Ku, J., Kurita, N. R., Lage, C. S., Lambert, R., Lange, T., Langton, J. B., Le Guillou, L., Levine, D., Liang, M., Lim, K.-T., Lintott, C. J., Long, K. E., Lopez, M., Lotz, P. J., Lupton, R. H., Lust, N. B., MacArthur, L. A., Mahabal, A., Mandelbaum, R., Markiewicz, T. W., Marsh, D. S., Marshall, P. J., Marshall, S., May, M., McKercher, R., McQueen, M., Meyers, J., Migliore, M., Miller, M., and Mills, D. J., "LSST: From Science Drivers to Reference Design and Anticipated Data Products," *ApJ* **873**, 111 (March 2019). DOI: https://doi.org/10.3847/1538-4357/ab042c. 1

[2] Graham, M. L., "Guidelines for User Support with the Rubin Community Forum," Technical Note RTN-097, NSF-DOE Vera C. Rubin Observatory (March 2026). https://rtn-097.lsst.io/. 2

[3] Graham, M. L., "Rubin Observatory's Approach to Providing Sustainable Scientific User Support at Scale," Technical Note RTN-121, NSF-DOE Vera C. Rubin Observatory (May 2026). https://rtn-121.lsst.io/. 2

[4] Rubin Observatory Science Pipelines Developers, "The LSST Science Pipelines Software: Optical Survey Pipeline Reduction and Analysis Environment," Project Science Technical Note PSTN-019, NSF-DOE Vera C. Rubin Observatory (December 2025). https://pstn-019.lsst.io/. 2

[5] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D., "Retrieval-augmented generation for knowledge-intensive nlp tasks," in [*Advances in Neural Information Processing Systems*], Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., eds., **33**, 9459–9474, Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf. 2

[6] Hsia, D., "RAG vs finetuning: which is the best tool to boost your LLM application?." Towards Data Science (2023). https://towardsdatascience.com/rag-vs-finetuning-which-is-the-best-tool-to-boost-your-llm-application-94654b1eaba7. 2

[7] Guy, L. P., AlSayyad, Y., Bechtol, K., Bellm, E. C., Blum, R. D., Dubois-Felsmann, G. P., Economou, F., Graham, M. L., Ivezić, Ž., Lupton, R. H., Marshall, P., O'Mullane, W., Slater, C. T., and Strauss, M. A., "Rubin Observatory Plans for an Early Science Program," Technical Note RTN-011, NSF-DOE Vera C. Rubin Observatory (April 2026). https://rtn-011.lsst.io/. 3

[8] Vera C. Rubin Observatory Team, "The Vera C. Rubin Observatory Data Preview 1," Technical Note RTN-095, NSF-DOE Vera C. Rubin Observatory (May 2026). https://rtn-095.lsst.io/. 3

[9] Jurić, M., Axelrod, T. S., Becker, A. C., Becla, J., Bellm, E. C., Bosch, J. F., Ciardi, D. R., Connolly, A. J., Dubois-Felsmann, G. P., Economou, F., Freemon, M. D., Johnson, M. W. G., Gill, R. K., Graham, M. L., Guy, L. P., Ivezić, Ž., Jenness, T., Kantor, J. P., Krughoff, K. S., Lim, K.-T., Lupton, R. H., Mueller, F., Nidever, D. L., O'Mullane, W., Patterson, M. T., Petravick, D. L., Shaw, R. A., Slater, C. T., Strauss, M. A., Swinbank, J. D., Tyson, J. A., Wood-Vasey, W. M., and Wu, X., "Data Products Definition Document," Systems Engineering Controlled Document LSE-163, NSF-DOE Vera C. Rubin Observatory (July 2023). https://lse-163.lsst.io/. 3

[10] Jenness, T., Bosch, J. F., Salnikov, A., Lust, N. B., Pease, N. M., Gower, M., Kowalik, M., Dubois-Felsmann, G. P., Mueller, F., and Schellart, P., "The Vera C. Rubin Observatory Data Butler and pipeline execution system," in [*Software and Cyberinfrastructure for Astronomy VII*], *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **12189**, 1218911 (August 2022). DOI: https://doi.org/10.1117/12.2629569. 3

[11] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H., "Retrieval-augmented generation for large language models: a survey," *arXiv preprint arXiv:2312.10997* (2024). DOI: https://doi.org/10.48550/arXiv.2312.10997. 4

[12] Economou, F. and Allbery, R., "Guidelines for gated updates for SQuaRE services (including Science Platform)," SQuaRE Technical Note SQR-056, NSF-DOE Vera C. Rubin Observatory (September 2021). https://sqr-056.lsst.io/. 5

[13] O'Mullane, W., Economou, F., Huang, F., Speck, D., Chiang, H., Graham, M. L., Allbery, R., Banek, C., Sick, J., Thornton, A. J., Masciarelli, J., Lim, K., Mueller, F., Padolsi, S., Jenness, T., Krughoff, K. S., Gower, M., Guy, L. P., and Dubois-Felsmann, G. P., "Rubin Science Platform on Google: the story so far," in [*Astromical Data Analysis Software and Systems XXXI*], Hugo, B. V., Van Rooyen, R., and Smirnov, O. M., eds., *Astronomical Society of the Pacific Conference Series* **535**, 227 (May 2024). DOI: https://doi.org/10.48550/arXiv.2111.15030. 6

[14] Liu, J. et al., "Recursive retriever and node references." LlamaIndex Documentation (2024). https://docs.llamaindex.ai/en/stable/examples/retrievers/recursive_retriever_nodes/. 8

[15] Yang, S., "Advanced RAG 01: small-to-big retrieval." Towards Data Science (2023). https://medium.com/data-science/advanced-rag-01-small-to-big-retrieval-172181b396d4. 8

[16] Kamradt, G., "5 levels of text splitting," (2024). https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb. 8

[17] Smith, B. and Troynikov, A., "Evaluating chunking strategies for retrieval," (2024). https://research.trychroma.com/evaluating-chunking. 8

[18] Chase, H., "LangChain: Building applications with LLMs through composability," (2023). https://www.langchain.com. 8

[19] Es, S., James, J., Espinosa-Anke, L., and Schockaert, S., "RAGAS: automated evaluation of retrieval augmented generation," *arXiv preprint arXiv:2309.15217* (2023). DOI: https://doi.org/10.48550/arXiv.2309.15217. 10