Building, Serving, and Growing a Conversational AI Assistant for Enterprise

Sumit Bhatia
Bhumir Jhaveri
Vidit Bhatia
Uttaran Bhattacharya
Victor Soares Bursztyn
Liqun Chen
Xiang Chen
Sally Fang
Horia Galatanu
Manas Garg
Shaddy Garg
Rachel Hanessian
Alex Hodorogea
Shun Jiang
Adobe
San Jose, USA

Nishant Kapoor
Yongsung Kim
Eunyee Koh
Namita Krishnan
Yunyao Li*
Zifan Liu
Akash Maharaj
Tung Mai
Saayan Mitra
Vaishnavi Muppala
Ramasuri Narayanam
Soumyabrata Pal
Kun Qian*
yunyaol@adobe.com
Adobe
San Jose, USA

Sajjadur Rahman
Ken Russell
Bikas Saha
Siddhartha Sahai
Som Satapathy
Rohan Saxena
Sai Sree Harsha
Anirudh Sureshan
Shashank Tandon
Mehrab Tanjim
Saurabh Tripathy
Anirudh Verma
Fei Wu
Tong Yu
Adobe
San Jose, USA

Abstract

Enterprises increasingly seek conversational AI systems that can reason over both structured and unstructured knowledge to enhance productivity and decision-making. Building such systems requires integrating heterogeneous data, grounding responses in domain-specific context, ensuring compliance and explainability, and maintaining quality through continual improvement. We present a unified platform for building and evolving enterprise conversational AI assistants that integrates structured data reasoning over relational schemas with retrieval-augmented generation over unstructured documentation, enabling accurate and verifiable responses. To ensure extensibility, the platform incorporates modular onboarding workflows that allow distributed teams to safely add new data domains and use cases. A comprehensive evaluation framework connects human annotation, automated metrics, and severity-based diagnostics to drive continuous quality improvements. Together, these components form a practical, generalizable blueprint for enterprise-grade conversational AI—grounded, auditable, and continuously improving through interaction.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**.

Keywords

Artificial Intelligence, Generative AI, AI Assistant

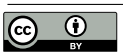
ACM Reference Format:
Sumit Bhatia, Bhumir Jhaveri, Vidit Bhatia, Uttaran Bhattacharya, Victor Soares Bursztyn, Liqun Chen, Xiang Chen, Sally Fang, Horia Galatanu, Manas Garg, Shaddy Garg, Rachel Hanessian, Alex Hodorogea, Shun Jiang, Nishant Kapoor, Yongsung Kim, Eunyee Koh, Namita Krishnan, Yunyao Li*, Zifan Liu, Akash Maharaj, Tung Mai, Saayan Mitra, Vaishnavi Muppala, Ramasuri Narayanam, Soumyabrata Pal, Kun Qian, Sajjadur Rahman, Ken Russell, Bikas Saha, Siddhartha Sahai, Som Satapathy, Rohan Saxena, Sai Sree Harsha, Anirudh Sureshan, Shashank Tandon, Mehrab Tanjim, Saurabh Tripathy, Anirudh Verma, Fei Wu, and Tong Yu. 2026. Building, Serving, and Growing a Conversational AI Assistant for Enterprise. In *Companion of the International Conference on Management of Data (SIGMOD Companion '26), May 31-June 05, 2026, Bengaluru, India.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3788853.3801880

1 Introduction

Generative AI Assistants for enterprises are designed to help their users find information, access and understand complex data, manage tasks, and automate operations. They hold great promise in significantly improving the productivity of their users, lowering the barrier-to-entry, drastically increasing product adoption, transformative amplification of creativity, and the delivery of better customer and employee experiences [4, 9]. In fact, much of the promise is rapidly becoming a reality in areas such as code development and customer support [1, 12].

The design desiderata for building an AI Assistant for enterprise include: (1) **Quality** - the AI Assistant needs to provide high-quality responses that its users can reply upon to assist their daily work; (2) **Trustworthiness** - the AI Assistant needs to build trust with its users. The users should be able to easily interpret and verify Assistant responses; (3) **Compliance** - the AI Assistant needs to give

*Corresponding author



SIGMOD Companion '26, May 31-June 05, 2026, Bengaluru, India

ACM ISBN 979-8-4007-2450-3/2026/05
https://doi.org/10.1145/3788853.3801880

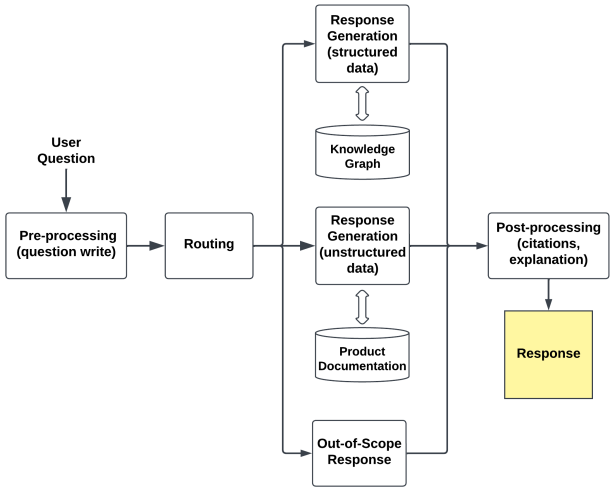


Figure 1: Overview of a Conversational AI Assistant for Enterprise

enterprises visibility and control of their data and ensure compliance, including how to safeguard data privacy, such as supporting Role Based Access Control (RBAC) constraints, and compliance with security laws; (4) **Usefulness** - the AI Assistant needs to support common use cases important to its users, including supporting different types of data, models and taking rich domain knowledge into consideration; and (5) **Usability** - the AI Assistant needs to be easy to use for a wide range of user personas, including helping its users to easily discover and utilize its current capabilities while minimizing any possible frustration.

Large language models (LLMs) provide the foundational technology behind many AI assistants, enabling them to understand and generate human-like language in a much more precise way than ever before. Even with the power of LLMs, many unique requirements and challenges in building, serving, and growing an AI Assistant for enterprise remain, including:

- *Complexity in underlying data.* Data underlying such an AI Assistant often includes enterprise proprietary data and knowledge and customer-specific data. Appropriate data structures, specifically semantic indexes for unstructured data and a relational knowledge data for structured data and semi-structured data, have to be designed and built to abstract away the complexity of the original raw data and enable efficient and effective retrieval for retrieval augmented generation at runtime.
- *Need for domain knowledge*: Domain knowledge is essential for grounding AI Assistant responses in enterprise-specific contexts. Unlike generic LLM applications, an enterprise AI Assistant must accurately interpret domain-specific terminology, entities, and relationships that are unique to an organization's products, workflows, and user intents. This requires multi-layered grounding — including schema-level alignment with enterprise data models, semantic alignment with business ontologies, and fine-tuning or prompting strategies that reflect real-world customer use cases. Capturing, representing, and dynamically updating domain knowledge remains an ongoing challenge due to the evolving nature of enterprise ecosystems.
- *Complexity of model development and improvement*: A wide range of models including off-the-shelf public and open-source LLMs, in-house task-specific models fine-tuned or trained for various downstream tasks, and traditional predictive models need to be employed across various components of an AI Assistant. These models differ in capabilities, latency, and data constraints, requiring an orchestration layer that can dynamically select and compose model outputs based on use case requirements and cost-performance trade-offs. Continuous improvement of these models requires automated evaluation pipelines, dataset curation strategies, and closed-loop feedback mechanisms from user interactions to inform model improvement.
- *Stringent requirements on quality*: Additional guardrails are required to measure the quality of the response and alert the user for potential quality issues (such as hallucination). These include automated quality estimation modules that assess factuality, relevance, and fluency; grounding audits to ensure retrieved evidence supports the generated output; and evaluation frameworks that measure effectiveness across multiple enterprise dimensions such as coverage, correctness, and interpretability.
- *Needs for growth*: Building an effective an AI Assistant is not a one-time engineering effort but a continuous process of growth across capabilities, coverage, and personalization. Sustaining this growth requires a robust life-cycle management infrastructure that supports experimentation, safe deployment, regression monitoring, and systematic knowledge updates.
- *Complexity in governance*: Governance encompasses ensuring compliance, visibility, and control over the full life-cycle of an AI Assistant's operation. This includes enforcing enterprise security controls such as RBAC, auditing and explainability of LLM outputs, maintaining lineage and traceability for data and prompts, and supporting enterprise-wide observability. Governance also requires policy-driven constraints to ensure responsible AI practices and alignment with regional data protection laws while preserving innovation velocity.

This paper introduces a next-generation platform for the building, serving and growing conversational AI Assistants for Enterprise, such as the AI Assistant in Adobe Experience Platform. The paper describes the system considerations and design decisions we followed to build our platform. Section 2 introduces the enterprise knowledge graph, which abstracts structured data for verifiable reasoning. Section 3 presents the document ingestion and indexing pipeline for organizing unstructured content. Section 4 outlines the conversational experience layer for multi-turn interaction management and intent routing. Section 5 and Section 6 describe the question answering pipelines grounded over documentation and the knowledge graph. Section 7 discusses the serving infrastructure, emphasizing deployment, access control, and multi-region scalability. Section 8 details the models for supporting extensibility and controlled growth of the AI Assistant, followed by Section 9 on evaluation and continual improvement.

2 Enterprise Knowledge Graph

The enterprise Knowledge Graph (KG) is responsible for abstracting away the complexity of the underlying structured enterprise data and providing the necessary business context to enable accurate,

explainable, and verifiable AI Assistant responses. The key challenges and considerations for building an enterprise-scale knowledge graph include:

- Different systems in an enterprise product portfolio provide different access mechanisms to fetch data, leading to the need to support multiple data source integrations.
- Every source contains a list of separate entities (such as customer segments or journeys), with each entity representing a collection of data. There is a need to build meaningful relationships between these in an automated fashion using appropriate business logic. Modeling the relationships across entities is crucial for answering complex cross-entity questions (for example, what are the segments connected to a given set journeys?).
- The schema for the KG needs to be designed in a way so that is easy for LLMs to understand and build queries for.
- The KG needs to provide a low-latency query layer for access control enforcement.
- The query language supported by the KG needs to be expressive and LLM-friendly so that users can ask a wide range of questions against the KG and get the correct results.

With the above considerations in mind, the KG is designed to store data for various entities, along with their relationships in a relational database. SQL is chosen as the query language for its expressivity, flexibility and that fact that LLMs have already demonstrating strong performance in text-to-SQL [8] generation. The details around key components of the KG are shared in the following sections.

2.1 Knowledge Graph Schema

The KG schema is based on star schema, an approach widely used in data warehousing. A single schema containing connected entities helps construct SQL queries for questions related to multiple entities. The KG includes three types of data tables:

- *DIM Tables*: DIM (Dimension) Tables store metadata about entities such as entity name, entity creation details, etc. Each row in this table represents a unique entity. It could store raw or transformed data about the entity. This table is used as a reference for other tables and as ground truth for other entities. These tables are named as $hkg\_dim\_\{entity\}$.
- *FACT Tables*: Any insights or metrics related to an entity are stored as FACT tables. Multiple rows can exist for a single entity such as historical data for a characteristic of an entity. These tables are connected to the corresponding dimension tables via foreign keys. These tables are named as $hkg\_fact\_\{fact\}$.
- *Bridge Tables*: Bridge tables exist as bridges between different tables to represent many-many keys. These tables are essential for answering questions about lineage across different entities. These tables are named as $hkg\_br\_\{entity1\}\_\{entity2\}$.

For easier generation, the schema is not strictly normalized and we do allow denormalization in some cases where it can help LLMs accurately generate queries. For example, in certain FACT tables that always require a join with the corresponding DIM tables, we do allow some columns like the entity name to be duplicated across both tables. While generating text-to-SQL, this kind of schema helps LLMs analyze user questions and use the appropriate table types in the SQL queries generated.

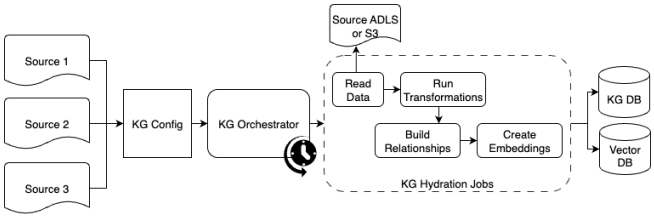


Figure 2: Knowledge Graph Hydration Process

2.2 Knowledge Graph Hydration

The KG hydration process runs on a regular cadence to pull data from various internal sources, transform, find relationships or lineages, and finally store data in the relational tables. As mentioned earlier, internal systems store data in different storage with different data formats. The challenge is to fetch data lying in different formats in different sources and support different pulling mechanisms. Thus, we build a KG hydration process to support various source ingestion patterns such as API, cache and cloud storage. To simplify hydration, a common integration pattern is introduced, using cloud storage as the standardized landing zone for all new sources. Internal systems are required to publish their data to this shared cloud storage and register the corresponding data locations in the KG configuration.

The overall high level KG hydration process is shown in Figure 2. The hydration process includes the following steps:

- *Source Registration*: Entities and their data locations are registered in the KG before the hydration process begins.
- *Data Pull and Transformation*: Once a new source cloud storage data location is registered, the hydration process retrieves data from the specified locations. If necessary, transformations are applied during storage in SQL tables.
- *Relationship Handling*: Relationships between entities must also be registered with an internal service at the time of their creation. These relationships are initially stored as a graph. While constructing the bridge tables for different entities, this relationship graph is recursively parsed using Breadth-First-Search to expand all possible relationships. These fully traversed relationships are then stored in the bridge tables to store relationships between entities. Figure 3 details an example on how we build these bridge tables from the relationship graph.
- *Embedding Creation*: Apart from relationships, the embeddings for the entity names and descriptions are also computed in this step and stored in a vector database for use by the Entity Linking Service as explained in section 6.

2.3 Querying the Knowledge Graph

The KG provides a query layer that enables an AI assistant to run low-latency queries on relational tables. Enforcing access control is a key aspect of this service to ensure secure data access.

One of the key requirements for an AI assistant is to ensure low latency. This poses a key challenge: achieving low latency for any (potentially complex) query generated by the LLM. To address this,

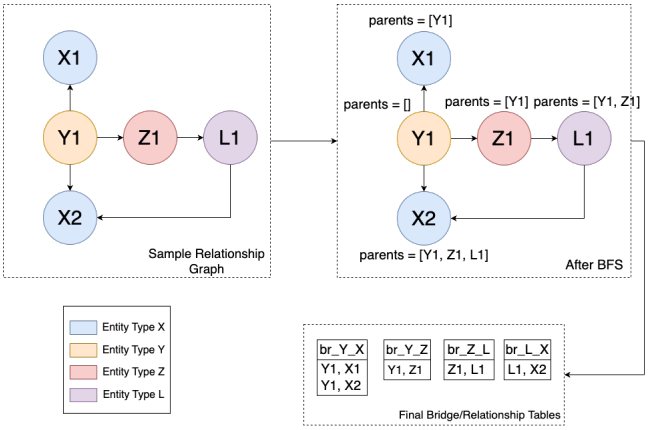


Figure 3: Extracting relationships from the Knowledge Graph

we adopt a modern database solution to efficiently handle such complex queries. Additionally, guardrails are introduced to limit the size of large result sets, further reducing overall response time.

Access controls are applied at the entity (table) level, including restrictions on the specific rows or columns within the tables. Only users with legitimate permissions are allowed to run queries, retrieve specific rows or view certain columns. Access controls are defined and stored by an access control service. The KG uses an AST parser to parse SQL queries, extract access control conditions from the internal service and enforce the appropriate restrictions by injecting conditional clauses appropriately.

3 Document Ingestion and Indexing

A critical capability of an enterprise AI Assistant is the ability to effectively ingest and index unstructured data. This component is responsible for transforming diverse enterprise content—such as documentation, guides, and help resources—into structured representations that can be efficiently searched and retrieved at runtime. Designing a document ingestion and indexing system for enterprise -scale applications involves several key challenges:

- **Heterogeneous document formats:** Enterprise content exists in multiple formats, such as markdown, PDF, and HTML, each requiring distinct parsing strategies.
- **Domain organization and versioning:** Enterprises operate across multiple product areas and knowledge domains, necessitating isolation between domains, support for versioning, and mechanisms to safely manage incremental updates.
- **Scalability and freshness:** The pipeline must handle large-scale ingestion and frequent updates while ensuring index consistency, freshness, and low-latency retrieval.
- **Configurability and adaptability:** Different document types, structures, and downstream use cases demand configurable chunking, embedding, and indexing strategies to optimize retrieval relevance.
- **Performance and reliability:** The system must ensure efficient vector search at runtime while meeting enterprise requirements for latency, fault tolerance.

To address these challenges, we developed a modular, asynchronous ingestion and indexing pipeline optimized for scalability, configurability, and operational robustness.

All content ingested into the system originates from verified product documentation and other enterprise-approved sources. Documents are organized into different document *collections*, which represent distinct product or domain areas (e.g., Customer Journey, Segments, or Analytics). Each collection is versioned, allowing for safe updates, rollbacks, and controlled deployment of new documentation versions across environments.

The ingestion process is implemented as an asynchronous queue-based system that decouples document upload, processing, and indexing. Documents are first uploaded to cloud storage and then processed through a configurable ingestion workflow. First, Documents are parsed and divided into semantically meaningful segments using configurable chunking strategies, such as fixed token windows or heading-based segmentation. Each chunk is transformed into a dense vector representation using a configurable text embedding model, supporting both open-source and fine-tuned in-house encoders. The embeddings, along with metadata (document ID, version, collection, and section hierarchy), are stored in a vector database for retrieval.

We employ a PostgreSQL-based vector database (pgvector) to store and manage document embeddings. This design combines efficient similarity search with transactional integrity, ensuring both performance and data consistency. Optimizations such as batched inserts, hybrid similarity metrics, and metadata-based filtering enable low-latency, high-throughput retrieval.

The ingestion and indexing pipeline is designed to evolve alongside enterprise data needs. New document types, embedding models, or chunking strategies can be introduced without disrupting existing collections. Comprehensive monitoring captures ingestion latency, index growth, and embedding quality metrics, enabling continuous assessment and optimization.

4 Conversation Experience

The conversational experience in the AI Assistant is designed to support complex, multi-turn interactions between a user and the AI Assistant. The key design considerations when building the conversation flow include;

- A user input may be under-specified or contain typos and grammatical errors. The AI Assistant should reduce user effort by automatically resolving such issues as much as possible and ask user input only when needed.
- A user is unlikely to be fully aware of AI Assistant's capabilities and hence will likely ask questions not yet supported by the AI Assistant. The AI Assistant should gracefully handle such situation and help the user to better understand and leverage the AI Assistant's capabilities.
- No model will work correctly all the time. Thus, it is important to reduce user frustration and improve AI Assistant response quality by allowing a user to recover from errors.

4.1 Question Rewrite and Ambiguity Detection

A key component of conversational experience which ensures accurate interpretation of the user input is the **Need-to-Rewrite** classifier, which detects **syntactical** and **pragmatic** ambiguities in the user's input.

Syntactical ambiguity refers to unclear phrasing or grammatical issues. For example: "Business Event" is an incomplete question that lacks sufficient context, allowing multiple interpretations. "what is a segmnet?" is a typographical error that may affect the system's ability to correctly parse the question. In such cases, the system can usually infer intent but may choose to rewrite the question for clarity and proper parsing.

Pragmatic ambiguity on the other hand, concerns uncertainties in the user's intent or context. For the user request, "Show me the latest one" it is unclear whether the user refers to the latest dataset, job, or schema. For "Delete this dataset" it is ambiguous if multiple datasets were discussed earlier.

Furthermore, not every question requires rewriting; in many cases, the user input may already be well-formed and clear enough to process without modification. For example, a question like "List all my datasets?" is clear and specific. As such, the Need-to-Rewrite classifier also plays a crucial role in ensuring that the system only rewrites queries when necessary, preventing unnecessary modifications that may lead to additional latency and potential errors.

Once the Need-to-Rewrite classifier detects ambiguity in a user input, it passes the input to the **Question-Rewrite** component. This LLM based component uses the context of the current user input, such as the conversation history, to automatically resolve ambiguities in the user input. For example, if a user first requests "Show me the dataset from last month" and later asks "What is the size of it?", the Need-to-Rewrite classifier first detects the existence of a vague pronoun "it" in the new question; then the Question-Rewrite component uses the conversation history to resolve "it" as referring to the dataset mentioned earlier.

In addition to resolving ambiguities and ensuring that the resulting rewritten question is fully specified, the Question-Rewrite component also corrects typographical errors, resolves references to ambiguous entities, while preserving important details such as "duplicate" or "by created date." The combination of the Need-to-Rewrite and Question-Rewrite components allows the AI Assistant to process and understand multi-turn interactions effectively by maintaining coherence and context and avoiding confusion from vague or under-specified queries.

4.2 Intent Routing

Once an user input is deemed as unambiguous by the Need-to-Rewrite classifier or clarified by the Question-Rewrite component, it is then classified by an **Intent-Routing** component into one of the following predefined intent categories.

Documentation Based Question Answering: This intent covers questions that can be answered by product documentation, including those about the AI Assistant itself. For example, when a user asks "What is the difference between a dataset and a dataflow?" it would be routed to the pipeline for documentation based question answering for an appropriate response (Section 5).

Knowledge Graph Based Question Answering: This intent pertains to questions related to customer-specific data. An example might be "What is the source account for this dataflow?" is then routed to the pipeline for KG based question answering for an appropriate response (Section 6).

Out-of-Scope: This intent covers all questions that fall outside AI Assistant's current capabilities such as technical support requests or questions completely unrelated to AI Assistant's scope such as inquiries about the weather. For an Out-of-Scope question, AI Assistant generates a response indicating that the questions is beyond its current capabilities.

4.3 End-to-End Conversational Flow

AI Assistant's conversational flow integrates the Need-to-Rewrite, Question-Rewrite, and Intent-Routing components to ensure accurate and efficient question processing.

Providing a good conversational experience is also important when a user asks questions that are not within the scope of AI Assistant. Specifically, when a user asks an Out-of-Scope question, it is crucial for AI Assistant to help the user understand the scope of the current capabilities of AI Assistant and ask an in-scope question. To do so, the AI Assistant provides a meaningful response to explain the scope of its capabilities for Out-of-Scope questions. It also provides suggested questions so that the user can ask an in-scope question with minimal effort by directly clicking on a suggested question. An example of this interaction can be seen in Figure 4, where a response is provided along with relevant in-scope suggestions for the user to explore. In addition, since Intent-Router could occasionally misclassify a question, the AI Assistant also allows the user to override the system decision by directly routing the question to a downstream agent, with the exception of input deemed as harmful or inappropriate. For instance, consider a scenario where a user asks, "What do you know about the destination 'HTTP API'?" However, the Intent-Router incorrectly classifies this as a documentation based question. Since "HTTP API" refers to an entity object in this context, the user intends to route the question to Operational Insights. To address this misclassification, the AI Assistant provides the user with the option to bypass the system's decision and directly route the question to the correct processing pipeline. The user can accomplish this by appending specific keywords which deterministically directs the question to the KG based question answering pipeline. This process is illustrated in Figure 5.

5 Documentation Based Question Answering

The documentation based question answering pipeline in the AI Assistant generates answers grounded in product documentation for user questions. Users can leverage a natural-language interface to engage in three primary modes of interaction: pointed learning, where they seek targeted explanations of specific concepts related to the product; open discovery, where they explore the features and capabilities of the product in a more exploratory manner; and troubleshooting support, where they address operational issues.

While LLMs combined with retrieval augmented generation (RAG) frameworks have shown significant promise for general-purpose question answering, applying these frameworks in enterprise settings introduces several challenges such as

- A domain-specific lexicon
- Dynamic and heterogeneous nature of documentation
- Lack of annotated data for model training and evaluation
- Ensuring that the claims in the AI Assistant's response can be verified and trusted

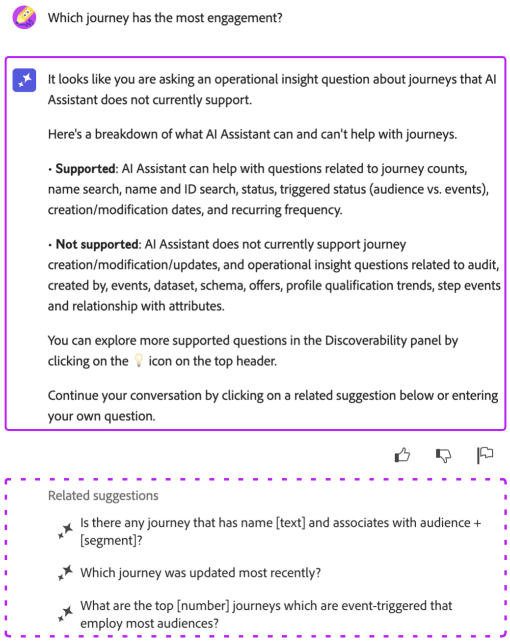


Figure 4: Example of an Out-of-Scope (OOS) response(solid line) with clickable in-scope suggestions(dashed line) to guide the user back to relevant topics.

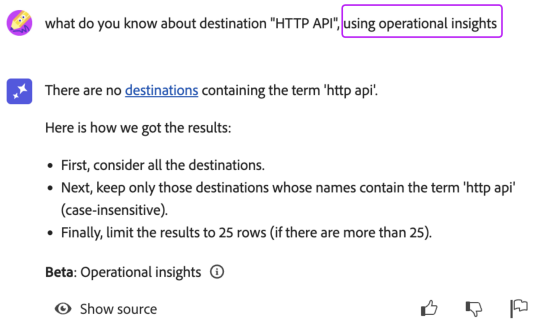


Figure 5: Example of a misclassification by the Intent Router, showing how the user can override the system and escalate the question to a human agent.

The documentation based question answering pipeline in the AI Assistant addresses these challenges by leveraging an orchestration of multiple components tailored to a domain-specific setting for delivering accurate and useful responses.

Figure 6 illustrates the high-level workflow and architecture of the documentation based question answering pipeline in the AI Assistant, which consists of four key components, (1) Product Classifier and Out-of-Scope Detection, (2) Document Retrieval, (3) Question Answering and (4) Attribution.

5.1 Product Classifier and Out-of-Scope Detection

It is common for enterprises to offer a portfolio of different products. When a question is received, classifying the question as being related to one or more product categories serves two key goals: (1) it helps in determining if the question is in the supported scope of the AI Assistant (2) determine the correct product document collection to use in the document retrieval step.

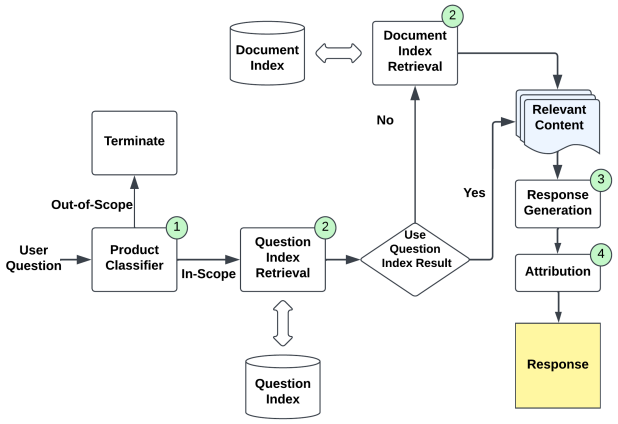


Figure 6: Documentation Based Question Answering Pipeline.

However, classifying user questions into product categories is challenging as user queries are often ambiguous and do not explicitly mention the product name. To address these challenges we build a product classifier by fine-tuning a Sentence-BERT model [14] using a dataset of (question, product category) pairs. Most pairs in the dataset are generated synthetically by prompting an LLM using passages from the product documentation and instructions to generate questions based on the passages. The dataset also includes a small number of pairs handcrafted by humans, particularly for products where documentation does not exist. Therefore, the product classifier can be easily adapted to support additional product categories using synthetically generated data from new product documentation

The product classifier is implemented as a multi-label text classifier model, which yields a probability distribution over product categories for a given question. A threshold is then used to determine the final product label(s) for that question. If the probabilities are lower than the threshold for all product categories, the pipeline halts further processing and displays a message to inform the user that the question is Out-of-Scope for the AI Assistant.

5.2 Document Retrieval

Once a question is determined to be in-scope and the correct document collection to use for a question is determined, the most relevant passages which can help answer the question are retrieved from the document collection. Unlike, traditional search algorithms represent data using discrete tokens or features, such as keywords, tags or metadata and rely on exact matches to retrieve relevant documents, we use vector search which represents data as dense vector embeddings and enables semantic similarity based search.

Document retrieval using a document-index. As described in Section 3, we use a vector database to store, manage and index the vector embeddings of passages from all documents in each document collection. Vector databases create indexes on vector embeddings that enable faster similarity searches between vectors. Given a question, we first compute a vector representation of the

question using a text embedding model. The database then calculates distances between the question vector and vectors stored in the index to return the top-k (e.g., 5) most similar vectors or nearest neighbors to the question vector. The passages corresponding to these top-k most similar vectors are passed on to the question answering module.

As noted in Section 3, enterprises often offer a range of different products, and can have multiple document collections corresponding to different products. For different document collections, different text embedding models can be used to ensure optimal document retrieval performance. For instance, in the AI Assistant for Adobe Experience Platform, while we use off-the-shelf pre-trained text-embedding models such as BGE [17] for some products, we also finetune a text-embedding model (Sentence-BERT [14])for some products (particulary those with very domain-specific vocabulary). However, generating a dataset for such a finetuning exercise is challenging to obtain as it requires subject matter experts (SMEs) to curate a large set a of (question, passage, relevance) triplets. Therefore, in addition to using a small set of human curated triplets, we also generate a larger set of such triplets by prompting an LLM (1) using passages from the documentation to generate questions based on the passed, (2) using a question and a passage to generate relevance labels.

Document retrieval using a question-index. In addition to the document vector store, we also build a vector store of questions with the associated relevant passages for each question as metadata. For any incoming user question, we first retrieve the most similar question(s) from this vector store of questions and pass on the corresponding correct passages to the question answering module, bypassing the document retrieval using the document index for questions with know relevant documentation. This step offers a mechanism to (1) rapidly and directly correct for errors in document-retrieval behavior, while training, evaluation and improvements to the text-embedding model for the document-index are in-progress, (2) guarantee correct document retrieval behavior for frequently-asked or high-value questions.

5.3 Question Answering

After the relevant passages from the product documentation are retrieved for a given question, the question and the retrieved passages are passed to the Question Answering module for answer generation. The Question Answering module prompts an LLM with (1) the user question, (2) the retrieved passages (3) task instructions and (4) in-context examples.

The task instructions consist of different types of guidelines that help control the response content and structure. Content guidelines ensure that the response is accurate and actionable, while preventing speculative, incomplete, or inappropriate responses, ensuring reliability and adherence to the provided context and documentation. Tone and style guidelines ensure that responses are friendly and informative and that the response is a good continuation of the conversation, fostering user engagement. Formatting and structure guidelines ensure responses are logically organized and easy to follow, using section headers, numbered points and bullets where necessary for clarity. It also includes instructions to generate potential follow-up questions which the users can ask based on the current question and the retrieved documents. These comprehensive guidelines enable the AI Assistant to generate coherent, trustworthy, engaging and useful responses.

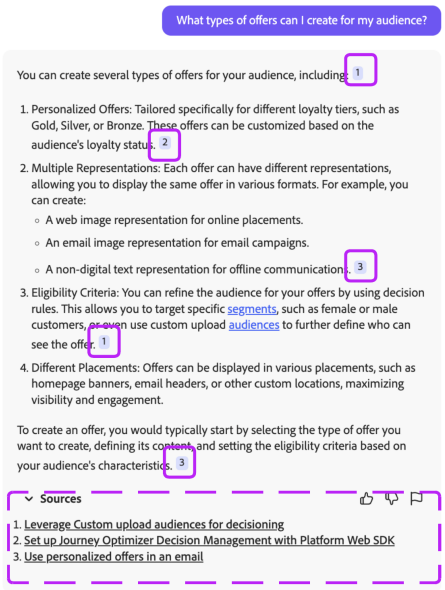


Figure 7: Example of inline references (solid line) and the Sources section (dashed line).

5.4 Attribution

To build user trust, we include a final attribution component as part of the documentation based question answering pipeline. The attribution component adds inline references in the LLM generated response to the passages retrieved and used to generate the response. Users can view these inline references in the "Sources" section as shown in Fig 7.

To add these references, we use a fine-tuned text-embedding model (described in Section 5.1) which compares the vector embedding of each sentence from the LLM generated response (V_R) and the vector embedding of each sentence from the passages retrieved and used in LLM prompt (V_P). If the cosine similarity between a given pair of vector embeddings is greater than a threshold, t, ($cosine\_similarity(V_R, V_P) > t$), we add an inline reference in the response to that passage. The attribution component offers users access to relevant documentation sources, ensuring that claims made in the LLM response can be checked for accuracy and trusted.

6 Knowledge Graph Based Question Answering

The knowledge graph based question answering in AI Assistant equips customers with daily, updated metrics and metadata usage statistics. Through a natural language interface, users can explore relationship mappings across data objects, such as audiences (customer segments), datasets, schemas, and journeys, gaining insights directly from the underlying knowledge graph.

The primary goal of the KG based question answering pipeline is to translate user natural language questions into SQL queries

(NL2SQL), enabling execution over databases and presenting results to users. While NL2SQL (a.k.a. text-to-SQL) has been extensively explored in academia[3, 6, 7, 13], there are fundamental differences between academic approaches and real-world industrial applications. The key challenges and considerations for building a KG based question answering pipeline for enterprise include:

- **Complex & Domain-specific Schemas:** Unlike the common-sense schemas in academic benchmarks such as SPIDER [18] and BIRD [5], industry datasets often feature proprietary, domain-specific, or business-specific schemas with closed-domain terminology.
- **Diverse & Ambiguous Question Formulations:** Industrial NL2SQL systems must handle a wide range of real-world natural language expressions, including ambiguous and context-dependent questions.
- **Evolving Database Schemas:** Frequent changes in database schemas in industrial settings require systems to be adaptable and resilient.

In order to build an industry strength NL2SQL system, we leverage a combination of advanced components to deliver accurate, reliable insights tailored to business needs. Figure 8 illustrates the high-level workflow and architecture of the KG based question answering pipeline, where there are three major RAG-based components and a standalone service: (1) Out-of-scope and In-scope detection (OOS/IS); (2) Natural language to SQL translation (NL2SQL); (3) Result Presentation & Explanation (Explanation); (4) Standalone Entity Linking Service (EL) When a natural language question (NLQ)

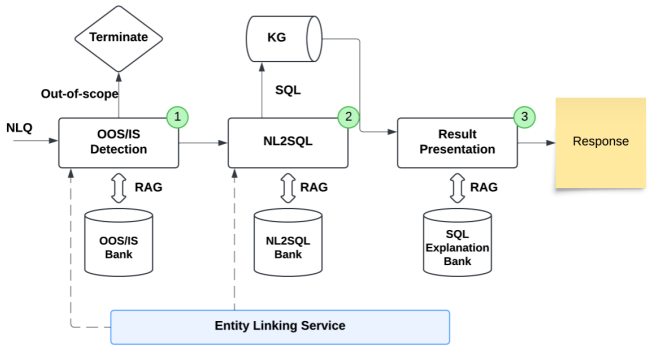


Figure 8: Knowledge Graph Based Question Answering Pipeline.

is received, it is processed through a pipeline of three sequential RAG-based components which ensure that the generated responses are both precise and reliable.

6.1 Out-of-scope and In-scope Detection

One major difference between open-domain AI assistants (like ChatGPT) and an AI Assistant for enterprise is that the latter operates as a closed-domain assistant, with a clearly defined scope of queries it can and is permitted to answer. The primary goal of the Out-of-Scope/In-Scope (OOS/IS) component is to determine whether an incoming question falls within the supported scope of the AI Assistant. If the question is out of scope, the pipeline halts further processing and does not generate SQL for result fetching. Instead, it displays a message to inform the customer that the question cannot be answered and provides guidance on how to rephrase the question to fall within the supported scope.

The OOS/IS component is based on a RAG approach. To enable LLM in-context learning, we create separate question banks for OOS and IS. The OOS question bank contains a collection of examples, where each example include:(1) an OOS natural language question `nlq` (2) a natural language explanation for why the NLQ is OOS, and (3) the main topic in the `nlq`. A similar structure is maintained in the IS question bank. Figure 9 shows an example from the OOS question bank. The explanation field is used to enable Chain-of-thought prompting to enhance LLM performance.

We embed the `nlq` fields of the OOS and IS examples into separate OOS and IS vector stores using a fine-tuned text embedding model (sentence-bert [15]). When a new NLQ Q enters the OOS/IS component, we turn Q into an vector V_Q using the same embedding model used for constructing the vector stores. We then retrieve the top-k (e.g., 5) semantically similar examples from both the OOS and IS vector stores. These examples are incorporated into the prompt, which is subsequently sent to the LLM for in-context learning.

```
{
  "nlq": "how many profiles were there in ABC audience 5
      days ago?",
  "explanation": "The question asks for the historical
      size of an audience, which is not support",
}
```

Figure 9: An example in the OOS question bank

6.2 NL2SQL

If a question is in-scope, it is passed to the NL2SQL module for SQL generation. The NL2SQL module is also RAG-based, whose prompt includes: (1) A system message that sets up the task context as translating natural language into SQL using a specific database syntax; (2) domain-specific instructions for SQL generation, providing relevant domain knowledge (such as domain-specific synonyms, limiting number of joins in generated SQL); (3) a subset of database tables from our KG that are relevant to the incoming NLQ; (4) few-shot demonstrations illustrating how NLQs are translated into corresponding SQLs over our KG schema.

Prompt content such as (1) a system message defining the task context and (2) domain-specific instructions are standard for most LLM-based tasks. We would like to emphasize (3) and (4) in this paper. First, rather than providing the full schema definition to the LLM, we include only a subset of our schema due to the complexity and large size of our KG schema. Additionally, the schema evolves rapidly, with biweekly changes in production driven by the rapid onboarding of new entities to the KG. Meanwhile, the actual queries from our customers are typically relatively short and focus on a small number of entities. To avoid providing the LLM with unnecessary information, we provide only a subset of the schema. This is achieved by creating a vector store of KG table embeddings, which is used at inference time to compare against the incoming NLQ. The top-k (e.g., 10) relevant tables are then retrieved. Similarly, for the few-shot NL2SQL demonstrations, we created a Text-to-SQL example bank containing examples of natural language queries mapped to their corresponding gold SQL queries over the KG. To facilitate efficient retrieval, we built a vector store with embeddings

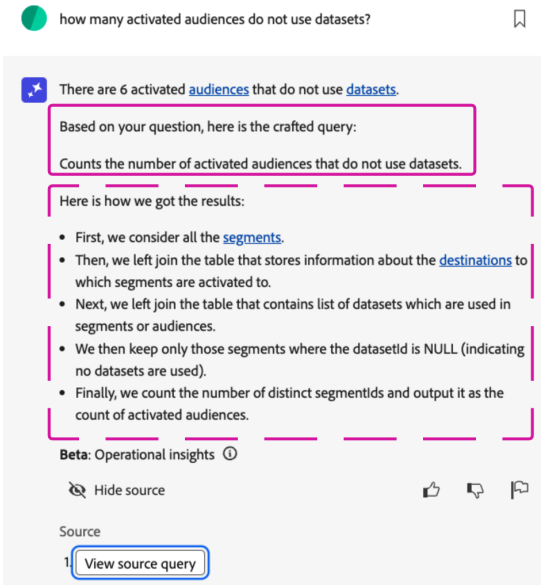


Figure 10: Question Interpretation (solid line) and Step-by-step breakdown (dashed line)

of the natural language parts of the Text-to-SQL examples. At inference time, the top-k examples are similar to the incoming NLQ are retrieved and used in prompt as few-shot examples.

Once a SQL is generated, as show in Figure 8, it is executed over the KG to retrieve the final results. Although not explicitly depicted in Figure 8, a retry loop is implemented to handle potential KG failures, particularly those arising from LLM hallucinations. Occasionally, the LLM may hallucinate columns or tables that do not exist, resulting in execution errors. In such cases, the retry mechanism is triggered, using the KG error message and the incorrect SQL in a retry prompt to request the LLM to correct the problem through a self-reflection process. If the issue cannot be resolved after three attempts, the process will terminate, and an error message will be sent to the users. If the retry is successful, the KG results will be retrieved and passed to the next component.

6.3 Final Result Explanation

To build user trust, instead of simply presenting the results to our customers, we add a final result explanation component to provide explanations. Explainability is critical not only for enhancing user trust and comprehension but also for enabling users to detect potential errors by identifying suspicious or inconsistent explanations.

Result Presentation. The primary objective of this task is to transform raw results from the KG query (initially provided as a data frame in a raw dictionary format) into a user-friendly format. This transformation ensures that the results are rendered clearly and naturally for users. For instance, consider the question: *'how many activated audiences do not use dataset'* as shown in Figure 10, the KG might return a dictionary like `{"COUNT(*)": 6}`, and the LLM would convert this to a more natural, user-friendly form, such as: "There are 6 activated audiences that do not use datasets".

We generate three distinct types of explanations, each tailored to address the needs of different user personas. These explanation types were identified using a data-driven approach [9].

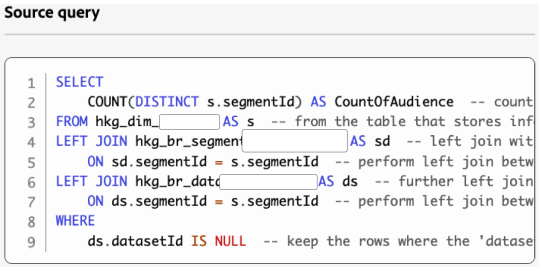


Figure 11: SQL inline comments (table names are masked out for confidentiality

Question Interpretation (illustrated in Figure 10 with solid bounding box) aims to provide a concise summary of actions taken to address the user's question. By masking the original question and focusing solely on the generated SQL, the LLM independently translates the SQL back into natural language. This reverse-engineering (SQL2NL) approach allows non-technical users to quickly verify results without needing to understand the SQL query generation process.

Step-by-step Breakdown (illustrated in Figure 10 with dashed bounding box) aims to provides a detailed step-by-step explanation of how the user question was processed. This breakdown is designed for users with a basic level of technical knowledge, enabling them to examine the steps taken by the LLM to interpret their question and retrieve the final results.

SQL Inline Comment (illustrated in Figure 11) aims to provide the generated SQL with inline comments, enabling users with strong technical knowledge to directly verify the queries. Since this explanation requires familiarity with SQL and our KG schemas, it is intended for advanced technical users, who are a minority among our customers. As an advanced feature, it is not by default displayed in the UI; users must click thethe 'View source query' button in the UI (see Figure 10) to view the SQL quesry in a pop-up window.

6.4 Entity Linking Service

As shown in Figure 8, a standalone and independent entity linking (EL) service, is utilized in two key stages: OOS/IS detection and NL2SQL. EL addresses the task of identifying entity mentions in user natural language queries (NLQ) and mapping them to corresponding KG objects. EL is crucial for question understanding as user questions are often ambiguous when referencing specific entities. For example, given user question *"Is there a product quality attribute?"*, where the user phrase *"product quality"* can actually refer ***aep.attr.Product_Quality***, and without EL, LLM does not have the context to generate the correct SQL, instead it may generate a SQL like the following, which clearly will result in an empty result response. With EL, we can ingest the matched KG object in-

```
SELECT * FROM ...
WHERE attribute_name LIKE '%product quality%';
```

formation into the NL2SQL prompt, and we also have EL-powered NL2SQL prompt instruction and examples so that the following correct SQL will be generated To be able to disambiguate both

```
SELECT * FROM ...
WHERE attribute_name in ('aep.attr.Product_Quality');
```

non-standard or approximate entity name mentions (e.g., "product

quality" ↔ "Product_Quality attribute") and semantically similar mentions (e.g., "location" ↔ "geography"), we designed a hybrid EL service that integrates string-based fuzzy matching with semantic-based matching.

7 Serving the AI Assistant

Serving an enterprise AI Assistant requires delivering retrieval-augmented generation at production scale while meeting strict guarantees on latency, access control, reliability, and governance. Unlike offline model training or ingestion workflows, serving operates in the user-facing critical path—where every request must execute efficiently, securely, and in compliance with enterprise policy. The main challenges arise from (1) maintaining low latency under heavy concurrent load, (2) enforcing fine-grained access control across tenants, (3) ensuring robustness and observability of a multi-component pipeline, and (4) optimizing cost and compliance across globally distributed deployments.

To ensure low latency and scalability, the serving layer in the AI Assistant is implemented as an asynchronous, event-driven pipeline that decouples retrieval, model inference, and response synthesis. Requests are streamed so that partial responses can be returned as soon as the LLM produces output, reducing perceived wait time. The infrastructure supports horizontal auto-scaling of inference workers and vector retrieval services across multiple geographic regions. Region-local routing minimizes round-trip latency and satisfies data-residency requirements, while shared control-plane orchestration ensures consistent behavior across deployments.

Access control is applied at multiple levels to ensure secure and compliant serving. At the outermost level, access to the AI Assistant itself is gated through service-level authentication and authorization policies that determine which users or applications can issue queries or view responses. Within the serving pipeline, additional access filters are applied at the data-source level to restrict retrieval from underlying enterprise assets.

The AI Assistant integrates fine-grained telemetry throughout the serving path to maintain system robustness and observability. Each request carries a request identifier that links logs from retrieval, orchestration, and inference components, enabling end-to-end trace reconstruction. Monitoring covers both infrastructure metrics (latency, throughput, failure rate) and semantic metrics (measuring response correctness for various)

Cost optimization in serving primarily centers around managing LLM inference and deployment strategy. Since LLM inference constitutes the dominant operational expense, the system employs a combination of model selection, workload partitioning, and deployment type optimization. For example, smaller models are used for simpler tasks such as classification, while larger models are reserved for complex tasks. On cloud providers such as Azure, deployment types directly influence serving economics. Provisioned throughput units (PTUs) are used for high-volume, latency-sensitive workloads that require guaranteed capacity and consistent response times. In contrast, pay-as-you-go or on-demand deployments are leveraged for bursty or experimental traffic where elasticity outweighs the need for reserved capacity. By dynamically routing requests across these deployment types based on traffic characteristics and model selection, the system achieves predictable performance while minimizing idle compute costs. Additionally, region-aware deployment ensures compliance with data-sovereignty requirements and reduces inter-region network charges.

Together, these design choices enable the AI Assistant to operate efficiently and securely at enterprise scale—balancing latency, reliability, and governance guarantees with sustainable cost and deployment efficiency across global regions.

8 Growing the AI Assistant

As the AI Assistant matures, expanding its coverage to new enterprise domains requires systematic mechanisms for growth. Scaling introduces two key requirements: (i) enabling internal teams to independently onboard new data sources or content, and (ii) ensuring that such additions preserve existing system accuracy and quality. To achieve this, we develop a self-service *collaboration model* that support controlled experimentation, validation, and deployment of new use cases.

The collaboration model for KG based question answering aims to enable seamless integration of additional data sources and relationships across business domains, and incorporate new attributes within existing tables. Achieving this expansion requires a systematic mechanism for extending the KG while maintaining reliability and accuracy. The key challenges in this context are:

- **Contextualization for new sources:** For each new data source, the LLM must be provided with sufficient schema context—such as entity relationships, lineage, and supported question templates—to correctly interpret and answer natural language queries.
- **Parallel development:** Multiple internal teams should be able to onboard, modify, and test schema or question updates in parallel without interfering with each other's work.
- **Backward compatibility:** The addition of new sources and question types should not degrade the accuracy, coverage, or latency of existing functionality.

To address these challenges, the collaboration model allows teams to independently onboard new data, evolve schemas, and expand question coverage through a standardized workflow. The model supports autonomous experimentation while ensuring that all updates undergo rigorous validation prior to deployment.

The process begins with *source registration*, where teams publish new datasets to a managed cloud location and update the KG configuration to include schema changes. This configuration specifies data transformations, lineage, and entity relationships, ensuring that new sources are seamlessly integrated into the existing KG schema. All schema updates are versioned to maintain consistency and traceability across experiments.

Following source registration, teams proceed with *onboarding representative (question, SQL) pairs*. Using the updated KG schema, they define representative sets of supported (in-scope) questions paired with SQL examples and document unsupported (out-of-scope) questions that cannot be answered with current data coverage. Maintaining both categories allows teams to define clear system boundaries and guide future schema evolution.

After schema and question updates are completed, the model enforces a comprehensive *validation and deployment process*. Validation includes testing paraphrased variations of new questions to assess robustness, performing regression tests to ensure existing

question-answer mappings remain stable, and conducting evaluations on the open-set (public validation dataset) and closed-set (private test dataset) to measure generalization. Only configurations meeting predefined accuracy thresholds are approved for release.

The collaboration model for documentation based question answering enables AI Assistant enables a systematic, repeatable process for onboarding new document sources, adapting models, and validating performance. The key challenges in this context are:

- **Expanding coverage:** Supporting incremental addition of new documentation and question sets, either by extending existing collections or creating new ones, while preserving index quality and retrieval performance.
- **Maintaining freshness:** Ensuring that frequent documentation updates are rapidly reflected in the retrieval and question indexes, and that classification and retrieval models remain aligned with evolving product terminology.
- **Rapid and robust evaluation:** Validating retrieval and generation performance efficiently using both automated and human evaluation before deployment.

To address these challenges, the collaboration model enables different teams to independently extend and validate the pipeline through standardized workflows for *content onboarding*, *index and model updates*, and *evaluation and deployment*.

Teams begin by integrating new documentation into the content store. Depending on the scope of the update, documents can be added to existing collections or organized into new ones corresponding to specific products or components. During ingestion, documents are parsed, chunked, and embedded into the vector database that powers semantic retrieval. Following ingestion, teams may add *question–document* (question–passage) pairs to the question index. As described in Section 5.2 each pair links a representative user question to the correct passage and is stored as a vector with metadata; during inference, a close match to an indexed question can be used to retrieve the linked passage directly, providing a fast corrective path for high-value or frequently asked queries.

If required, the *product classifier* is retrained to incorporate new product categories. The classifier, based on a fine-tuned Sentence-BERT model [14], is trained on (question, product category) pairs that are largely generated synthetically by prompting an LLM with passages from product documentation, complemented by a small number of human-curated examples. This design enables the classifier to be efficiently extended to support new products using data automatically derived from new documentation. If domain-specific terminology in new content affects retrieval accuracy, the *retrieval model* can also be fine-tuned to preserve quality.

After index and model updates are completed, the system enforces a comprehensive *validation and deployment process*. Validation includes testing to assess robustness, performing regression tests to ensure existing question–answer pairs remain stable, and conducting evaluations on the *open-set* (public development dataset) and *closed-set* (private holdout dataset) to measure generalization and accuracy on new sources. Both automated evaluations using LLM-based judges and human assessments are employed to measure factual consistency and response quality. Only updates meeting predefined quality thresholds are approved for deployment.

The collaboration models for the KG based and documentation based question answering pipelines enable multiple internal teams to safely and efficiently scale the AI Assistant's capabilities in a decentralized yet controlled manner.

9 Evaluation and Continual Improvement

Given the stringent requirements on quality in enterprise scenarios, the evaluation and continual improvement process for such an AI Assistant is of paramount importance. This section describes the challenges in developing an evaluation and improvement process, and our approach to addressing these challenges.

9.1 Design Considerations for Evaluation

Developing a trustworthy enterprise AI Assistant requires a structured evaluation framework that balances user-centric quality with system-level reliability [10]. The following design principles guide our approach to monitoring, benchmarking, and continual improvement across multiple teams.

Prioritize directly actionable metrics. Lagging indicators such as productivity or satisfaction evolve slowly; hence we track leading metrics—*correctness*, *factuality*, and *recovery rate*—that respond immediately to system changes and correlate with downstream user success. These are tied directly to production changes rather than abstract model scores.

User-aligned severity taxonomy. Not all errors equally impact user trust. We employ a three-level severity framework: (1) **Severity 0**—answers appear correct but are wrong (erosion of trust); (2) **Severity 1**—answers are visibly wrong and irrecoverable (frustration); and (3) **Severity 2**—answers are wrong but recoverable (annoyance). This taxonomy transforms subjective correctness into actionable metrics, allowing targeted improvements at both component and system levels.

Human–AI evaluation synergy. Despite challenges [16], human judgments remain best aligned with eventual user outcomes. Large-scale human annotation—guided by expert-validated decision trees—provides reliable ground truth, while validated automatic metrics (LLM-as-Judge, retrieval precision) enable scalable regression testing. Human evaluation is therefore prioritized but complemented by automated checks for coverage and speed.

Efficient sampling and annotation allocation. To maintain scalable quality assurance as usage grows, we apply coreset-based subsampling [2] to prioritize high-value samples for human review, reducing annotation costs by 20–30%. Non-experts annotate simple tasks; expert annotators resolve disagreements and conduct targeted error analysis.

Holistic, system-wide improvement. Quality emerges not only from model accuracy but also from retrieval quality, UX explainability, and user recovery affordances. Improvements are pursued vertically—across model, orchestration, and interface layers—rather than within isolated components [9].

Proactive and adversarial evaluation. We complement retrospective monitoring with proactive, regression-preventing evaluation. Adversarial testing programs enable error discovery where domain experts engage in exploratory testing to surface difficult failure modes [10]. Their findings populate continually refreshed holdout datasets for controlled, end-to-end comparisons.

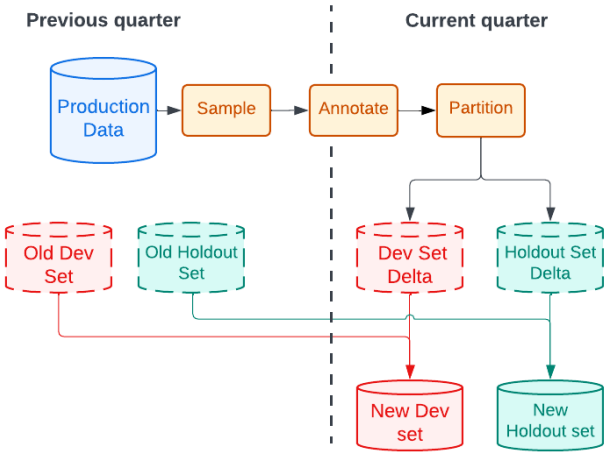


Figure 12: Creation of shared evaluation datasets on an ongoing basis, using sampling and human annotation of production traffic

Scalable, multi-stakeholder visibility. Executives, developers, and quality leads require distinct views of system performance. Shared dashboards visualize severity trends, regression deltas, and annotation throughput, supporting both macro (business) and micro (component) decision-making.

9.2 Evaluation Architecture

The evaluation architecture integrates automated logging, human annotation, and analytics dashboards to ensure efficiency and reproducibility across many development teams.

- **Shared datasets.** As shown in Figure 12, Production interactions are sampled and annotated to form quarterly *development* and *holdout* datasets, ensuring reproducible, like-for-like evaluation across releases.
- **Human annotation platform.** Non-expert annotators label masked production data; experts conduct fine-grained error analysis. Templates, guidelines, and pilot modules ensure inter-rater reliability.
- **Automated evaluation.** Validated LLM-as-Judge scorers provide fast regression signals, while statistical dashboards track error severity, coverage, and model drift.
- **Interactive dashboards.** Multi-persona dashboards enable drill-downs by severity, component, and customer cohort, linking regression metrics to underlying conversations.
- **Governance and templates.** Evaluation templates standardize metrics and reporting across teams, lowering onboarding friction for hundreds of developers.

9.3 Improvement Architecture

Continual improvement couples evaluation feedback with a structured enhancement loop (Figure 13).

- **Expert-in-the-loop analysis.** Domain experts triage errors to trace failure root causes—retrieval gaps, prompt defects, data schema issues—and recommend targeted fixes.
- **Systemic levers.** Improvements span prompt tuning, retrieval augmentation, synthetic data generation, explainability modules, and UX affordances enabling user recovery [11].

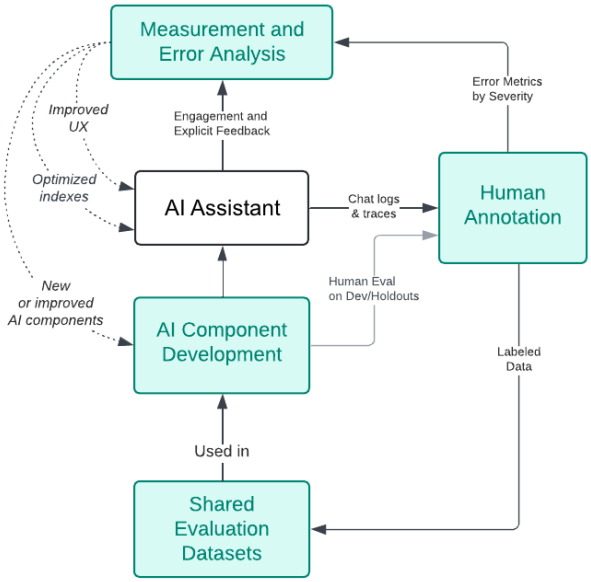


Figure 13: The continual improvement framework, emphasizing human annotation as a way of both generating labeled data to be used in shared evaluation datasets

- **Benchmark refresh and regression prevention.** Shared holdouts serve as baselines for new feature launches. Each launch replays historical queries to detect behavioral regressions before deployment.
- **Cross-team scalability.** Modular templates, data contracts, and dashboards make evaluation reproducible across independent components, sustaining quality at enterprise scale.

This integrated framework operationalizes human-centered, data-driven continual improvement.

10 Conclusion

We introduced a production-scale platform for conversational AI in enterprise environments, designed to unify structured analytics and unstructured knowledge retrieval under a single interaction and evaluation framework. By abstracting enterprise data into a knowledge graph for structured reasoning and maintaining dynamically updated document indexes for unstructured content, the platform enables accurate, explainable, and trustworthy responses to complex natural-language queries. The architecture emphasizes modularity and collaboration: its onboarding workflows allow teams to integrate new data sources and knowledge domains without compromising existing functionality, while standardized validation and regression testing ensure that quality scales alongside coverage. The evaluation and continual improvement framework links human feedback, automated scoring, and error analysis into a closed loop of measurement, diagnosis, and refinement. Looking forward, principles established in our platform—structured grounding, and human-centered evaluation—can inform the next generation of adaptive, multi-domain conversational systems capable of learning from interaction, maintaining compliance, and evolving with organizational needs.

References

[1] Erik Brynjolfsson, Danielle Li, and Lindsey Raymond. 2023. Generative AI at Work. *SSRN Electronic Journal* (2023). https://api.semanticscholar.org/CorpusID:258298324

[2] Trevor Campbell and Tamara Broderick. 2018. Bayesian coreset construction via greedy iterative geodesic ascent. In *International Conference on Machine Learning*. PMLR, 698–706.

[3] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. arXiv:2308.15363 [cs.DB] https://arxiv.org/abs/2308.15363

[4] Akit Kumar, M.S. Lakshmi Devi, and Jeffrey S. Saltz. 2023. Bridging the Gap in AI-Driven Workflows: The Case for Domain-Specific Generative Bots. In *2023 IEEE International Conference on Big Data (BigData)*. 2421–2430. https://doi.org/10.1109/BigData59044.2023.10386894

[5] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL] https://arxiv.org/abs/2305.03111

[6] Yunyao Li, Dragomir Radev, and Davood Rafiei. 2024. *Natural language interfaces to databases*. Springer.

[7] Yunyao Li and Davood Rafiei. 2018. *Natural language data management and interfaces*. Morgan & Claypool Publishers.

[8] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109* (2024).

[9] Akash Maharaj, Kun Qian, Uttaran Bhattacharya, Sally Fang, Horia Galatanu, Manas Garg, Rachel Hanessian, Nishant Kapoor, Ken Russell, Shivakumar Vaithyanathan, and Yunyao Li. 2024. Evaluation and Continual Improvement for an Enterprise AI Assistant. In *Proceedings of the Fifth Workshop on Data Science with Human-in-the-Loop (DaSH 2024)*, Eduard Dragut, Yunyao Li, Lucian Popa, Slobodan Vucetic, and Shashank Srivastava (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 17–24. https://doi.org/10.18653/v1/2024.dash-1.3

[10] Akash V. Maharaj, David Arbour, Daniel Lee, Uttaran Bhattacharya, Anup Rao, Austin Zane, Avi Feller, Kun Qian, and Yunyao Li. 2025. Evaluation and incident prevention in an enterprise AI assistant. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence (AAAI'25/IAAI'25/EAAI'25)*. AAAI Press, Article 3263, 7 pages. https://doi.org/10.1609/aaai.v39i28.35161

[11] Akash V Maharaj, David Arbour, Daniel Lee, Uttaran Bhattacharya, Anup Rao, Austin Zane, Avi Feller, Kun Qian, Sajjadur Rahman, and Yunyao Li. 2025. Evaluation and incident prevention in an enterprise AI assistant. *AI Magazine* 46, 3 (2025), e70028.

[12] Radu Miclaus, Arun Chandrasekaran, Ray Valdes, Mark Driver, and Eric Goodness. 2023. Generative AI Code Assistants Are Becoming Essential to Developer Experience.

[13] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions. arXiv:2208.13629 [cs.CL] https://arxiv.org/abs/2208.13629

[14] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[15] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL] https://arxiv.org/abs/1908.10084

[16] Chris van der Lee, Albert Gatt, Emiel van Miltenburg, and Emiel Krahmer. 2021. Human evaluation of automatically generated text: Current trends and best practice guidelines. *Computer Speech & Language* 67 (2021), 101151.

[17] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597 [cs.CL]

[18] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL] https://arxiv.org/abs/1809.08887