International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 7, Issue 2, 98-104, 2026
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I2P117

Original Article

Distributed RAG Architecture for Enterprise Knowledge QA on Databricks

Vamshi Krishna Malthummeda
Independent researcher, USA.

Received On: 25/02/2026 ***Revised On: 03/04/2026*** ***Accepted On: 10/04/2026*** ***Published On: 17/04/2026***

Abstract: *Organizations rely on centralized knowledge repositories like Salesforce Knowledge to support internal teams and external teams. The traditional keyword/rule-based search often doesn't provide accurate, up-to-date information at scale. This paper presents a unified data-centric and model-centric Retrieval-Augmented Generation (RAG) architecture implemented on Databricks using Apache Spark. The architecture integrates large-scale ETL processing with efficient model execution for vector embedding generation within a single distributed framework. The proposed solution employs open and modular design with separate components for data extraction, data cleansing, workflow orchestration, embedding generation, vector retrieval and LLM inference. In the proposed approach, knowledge articles are extracted by executing Salesforce Object Query Language (SOQL) queries against a Salesforce Knowledge object REST API endpoint and stored in a Databricks Lakehouse. The RAG pipeline employs stateful model reuse and fine-grained resource allocation coupled with concurrent multithreaded encoding within Databricks to perform large-scale vector embedding generation. The generated vector embeddings from the extracted content are stored in a vector database. At inference time, user queries from the chatbot are converted into vectors and matched against the vector database to find most relevant information chunks which are then sent along with prompts to an LLM to generate responses. This design ensures that answers are traceable to authoritative knowledge sources, reduces hallucinations, and supports continuous updates as knowledge articles evolve.The RAG pipeline implemented on a Databricks cluster demonstrated 100% CPU utilization with high execution efficiency on a vector embedding pipeline as shown in the experimentation. The proposed architecture provides fine-grained control over resource allocation, concurrency, and batching strategies, enabling enterprises to generate high quality embeddings while leveraging existing infrastructure without significant redesign.*

Keywords**:** *Databricks, Vector Databases, Llm, Ray, Rag, Chatbot, Vector Embeddings, Salesforce, Soql.*

1. Introduction

1.1. Emergence of Question Answering Chatbots

Question Answering (QA) chatbots are one of the key AI innovations that revolutionize customer service by automating interactions and providing instant support across various domains like healthcare, education and enterprise knowledge management. The rise of QA chatbots is the culmination of development in various fields like Natural Language Processing (NLP), Information Retrieval (IR) and Deep Learning.

1.2. Limitations of Rule-Based and Keyword-Based QA Systems

Rule-based and keyword-based QA systems face significant limitations primarily in scalability, adaptability, and language understanding. They rely on static predefined logic, making them rigid and unable to handle the complexity and nuance of human language or new information. The responses from legacy QA systems are robotic in nature and lack personalized, human-like tone that advanced AI systems can provide. They cannot improve their responses over time unless manually updated by a developer or domain expert.

1.3. Evolution of RAG based Generative QA Chatbots

The application of statistical NLP on top of information retrieval-based approaches to retrieve relevant documents and candidate answers using keyword matching and probabilistic ranking models was the first step towards addressing the limitations of the rule-based/keyword-based QA systems. The advent of deep learning models like BERT, GPT, and T5 enabled contextual understanding of the queries and passages of text.

The main drawback of the traditional retrieval-based QA systems is to select the responses from a predefined corpus, limiting their flexibility. To address this issue QA chatbots leveraging Large Language Models (LLMs) are developed which generate answers dynamically. Even the LLM-based

QA systems suffer from drawbacks like hallucinations and lack of factual groundings[3][7].

1.4. Scope and Contribution of this Paper

The scope of this paper is to design, implement, and analyze scalable enterprise architecture of a question answering (QA) Chabot. The Salesforce Knowledge is the source system on which the responses of the QA Chabot should be grounded.

This paper focuses on:

- Extracting knowledge content from Salesforce Knowledge objects using SOQL queries via REST APIs.
- Ingesting and transforming of knowledge content is done within the Databricks Lakehouse to enable unified batch and incremental processing.
- Transforming knowledge content into semantic vector embeddings using sentence transformer models, with Spark distributed processing system being leveraged to parallelize document chunking, embedding generation, and embedding storage in a vector store index for later similarity-based retrieval.
- Generating the response for question posed in chatbot by converting the question into vector embedding using Databricks foundation models and performing the similarity search against the vector store index.

This paper makes following contributions:

- Unlike monolithic AI solutions, this architecture decouples knowledge ingestion, transformation, and retrieval layers, allowing independent scaling, optimization, and governance of each component.
- This work contributes a highly performant embedding generation pipeline, and an empirical performance study demonstrating substantial efficiency gains within a Databricks-native environment.
- By leveraging open and extensible components, the proposed approach offers a cost-effective alternative to tightly integrated proprietary AI solutions like Salesforce Agent force while maintaining enterprise-grade performance and security.

2. Enterprise Knowledge Retrieval Rag Architecture

The proposed architecture integrates Salesforce Knowledge as the enterprise knowledge source, Databricks Lakehouse as the unified data and AI processing platform, and a vector index as the semantic retrieval layer to enable scalable and accurate question answering and retrieval-augmented generation (RAG). The architecture is designed to decouple data ingestion, transformation, embedding generation, and retrieval, thereby improving scalability, governance, and extensibility.

At a high level, knowledge articles are extracted from Salesforce using REST APIs, processed and stored in the Databricks Lakehouse, transformed into vector embeddings, and indexed for low-latency semantic search.

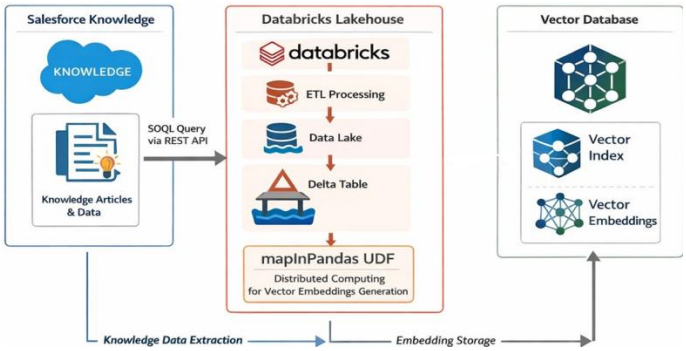

Fig 1: RAG Architecture for Enterprise Knowledge Retrieval

2.1. Salesforce Knowledge

Salesforce Knowledge is a powerful, centralized system within Salesforce for creating, managing, and sharing information as reusable "Knowledge Articles," serving as a self-service portal for customers (like FAQs, user guides) or internal resources (playbooks, training). It acts as a single source of truth for any organization preventing inconsistencies.

2.2. Salesforce Knowledge Ingestion Layer

Salesforce Knowledge articles data is ingested into Databricks Lakehouse ingestion job by querying standard **/services/data/vXX.X/query/** salesforce REST API endpoint by passing below SOQL queries as URL parameters, with spaces being replaced by the + sign (or %20) (If the record volume is high, then we need to use Salesforce BULK API)[2].

The SOQL query below is executed to get the metadata of all the Knowledge Articles

```
SELECT Id, Title, PublishStatus, ArticleType, LastModifiedDate
FROM KnowledgeArticleVersion
WHERE PublishStatus = 'Published'
ORDER BY LastModifiedDate DESC
```

For each of the articles fetched above the SOQL query below is executed to get the Article content and other details:

```
SELECT Id, Title, PublishStatus, VersionNumber, ArticleBody__c
FROM {ArticleType}__kav
WHERE PublishStatus = 'Published'
ORDER BY LastModifiedDate DESC
```

The extracted data is then serialized into an intermediate format (e.g., JSON) and pushed to the downstream ingestion pipeline.

2.3. Databricks Lakehouse Processing Layer

The Databricks Lakehouse acts as the central processing and storage layer that enables unified handling of raw, processed, and enriched knowledge data. The extracted data is upserted into delta tables for further processing.

Knowledge articles are cleaned, normalized, and segmented into semantically coherent chunks to optimize embedding quality and retrieval performance. Metadata such as article ID, category, language, and timestamps are preserved to support filtering and governance.

Each knowledge chunk is converted into a dense vector embedding using a sentence transformer model. The most compute intensive part of the RAG pipeline is the embedding generation of the knowledge article chunks.

The vector embedding generation process adopted by the proposed RAG pipeline utilizes the following process that is based on Apache Spark. In Apache Spark, the unit of parallel execution is a task, and tasks operate on partitions of data. In the proposed approach, the inital pre-processed knowledge chunks are loaded into a spark dataframe and subjected to repartitioning to ensure maximum task parallelism in the form of distributed parallelized model inference. The distributed task parallelism is triggered by calling the mapInPandas method on the repartitioned Spark Data Frame.

When the mapInPandas method is called, the following actions are performed:

- For each of the partitions a Spark task is created (the compute/cores associated with the task will be dictated by the spark.task.cpus Spark configuration setting. This is one of the resource allocation configurations available to allocate more cores for cpu bound tasks)
- Each Spark task launches a Python worker process. The models which generate embeddings are typically large and need to be loaded into memory after downloading from external location which is a time consuming and IO bound operation. The Python worker process plays an important role in providing model statefulness across multiple batches within the partition by keeping the model loaded within the Python worker process. The configuration parameter spark.python.reuse can keep the model loaded in the worker process to work on the next set of partitions. The model once downloaded by a Python worker process on a given worker node saves it to a CACHE FOLDER location on the worker node local disk. Other Python worker processes can load it from local disk once it is available which will increase the throughput by avoiding the multiple downloads
- The Python worker process then executes a Python native function which performs the vectorized operations on the batches of data belonging to the associated partition. Custom batching strategies can be applied by splitting the partition batches to sub batches within the function or at the cluster level by setting the right size for the park.sql.execution.arrow.maxRecordsPerBatch configuration setting. For higher concurrency, an implementation of fine-grained resource allocation and high CPU utilization can be achieved with time-slicing of cores associated with the Spark task by the creation of more threads than available cores using ThreadPoolExecutor within the python function)

The key capabilities of distributed parallel mode inferencing, resource allocation, custom batching strategies and effectively high CPU utilization are the hallmarks of a large-scale embedding generation pipeline and are implemented in the proposed RAG pipeline.

The above Databricks Spark-only pipeline performed better than pipeline implemented on Ray cluster atop the Databricks Spark cluster which can also provide the above key capabilities with little bit more flexibility and better resource accounting required for granular CPU allocation. The reason for better performance is because Spark is the native engine of the Databricks platform and Databricks is highly optimized for Spark task scheduling, leveraging data locality and efficient data exchange.
The comparison between Spark-only and the Ray cluster implementation is provided in the Experimental Setup section.

2.4. Vector Index Layer

A vector index is a specialized data structure used to store vectors of data for fast similarity searches. The vector index layer serves as the semantic retrieval engine for the system. An original text chunk and its vector embedding are paired, with the chunk (smaller text piece) being the meaningful content and the embedding (dense numerical array) the semantic representation, are both stored in a vector index (database/store) for fast similarity lookup.

Following are the key features of this layer:

- This layer allows number of relevant chunks to be retrieved based on the low-latency retrieval requirement.
- This layer allows metadata-based filtering when domain or access control constraints are applicable[6].
- Allows Incremental index updates to reflect knowledge content changes.

Databricks offers a fully integrated, built-in vector store solution called **Mosaic AI Vector Search**[1], but also allows integration with several popular third-party vector databases like Weaviate, Pinecone, ChromaDB, pgvector and others.

This layer enables semantic matching between user queries and relevant knowledge content, outperforming traditional keyword-based search approaches.

2.5. Query and Answer Retrieval Flow Layer

In this layer, the user is presented with a visual interface whih can be developed using front-end UI Python native libraries like Streamlit, Gradio and others, or can be developed using enterprise platforms from big tech companies like Google Dialogflow, Microsoft Bot

Framework, Amazon Lex, and IBM Watson Assistant. The visual interface allows users to interact with RAG system. The following steps are performed when a user submits a query:

- The query is converted into a vector embedding using the same embedding model as was used to construct the vector database.
- A similarity search is executed against the vector index to retrieve top-K relevant knowledge chunks.
- Retrieved content is ranked and assembled into a contextual prompt.
- The assembled context is passed to a large language model for answer generation using retrieval-augmented generation.
- The generated answer is presented to the users on UI[4].

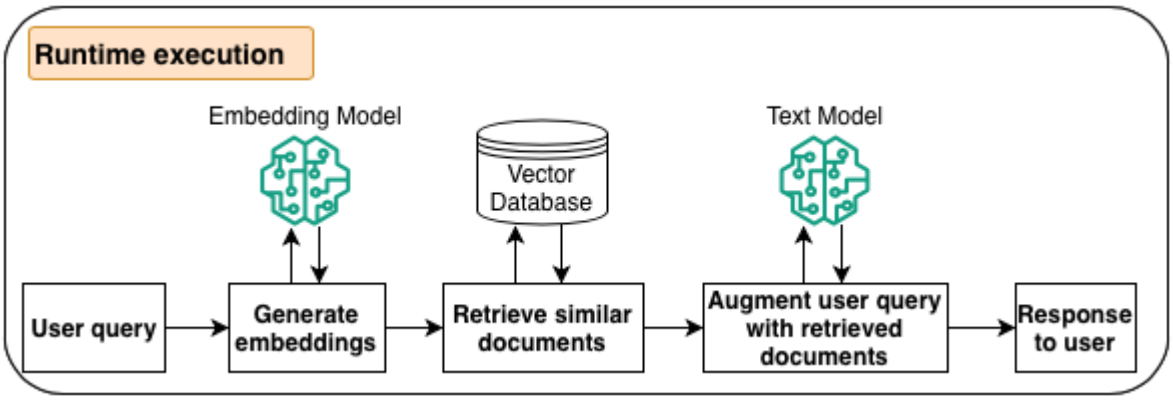

Fig 2: Question Answering System flow

3. Experimental Setup

3.1. Dataset

Below are the details of the dataset used for experimentation:

- Dataset Name: Financial Sentiment dataset
- Domain: Financial
- Source: Public dataset (downloaded from Kaggle)

Size

- Number of documents**: 15523**
- Average document length: 500 characters
- Size on Disk: 8.4 MB

Preprocessing:

- Chunking
- Repartitioning

3.2. Model Configuration

Embedding Model

- Model Name: SentenceTransformers(all-mpnet-base-v2)
- Embedding Dimension: 768
- Framework: PyTorch
- Hardware Acceleration: CPU
- Batch Size: 200

3.3. Retrieval Model

- Vector Database: Databricks Vector Search
- Index Type: Direct Access

3.4. System Infrastructure

Cluster Configuration

- Platform: Databricks
- Processing Engine: Apache Spark
- Worker Nodes (Executors): 10
- CPU per executor: 8 cores
- Memory per Executor: 32 GB

Runtime Configuration

- spark.task.cpus: 1
- max records per batch: 200
- spark.python.worker.reuse: true(default)
- repartition: 80

Ray Configuration

- num_cpus_worker_node: 8
- num_cpus_head_node:6
- max_worker_nodes: 10

3.5. Experimental Scenarios

Table 1: Embedding Performance Analysis: Spark mapInPandas vs Ray Actor Pipeline

Goal	Approach	Metrics
Embedding Execution Time	Spark mapInPandas	278.44 seconds
	Ray Actor Pipeline	343.02 seconds
Embedding CPU Utilization	Spark mapInPandas	95-100%
	Ray Actor Pipeline	90-95%

3.6. Spark Only Cluster

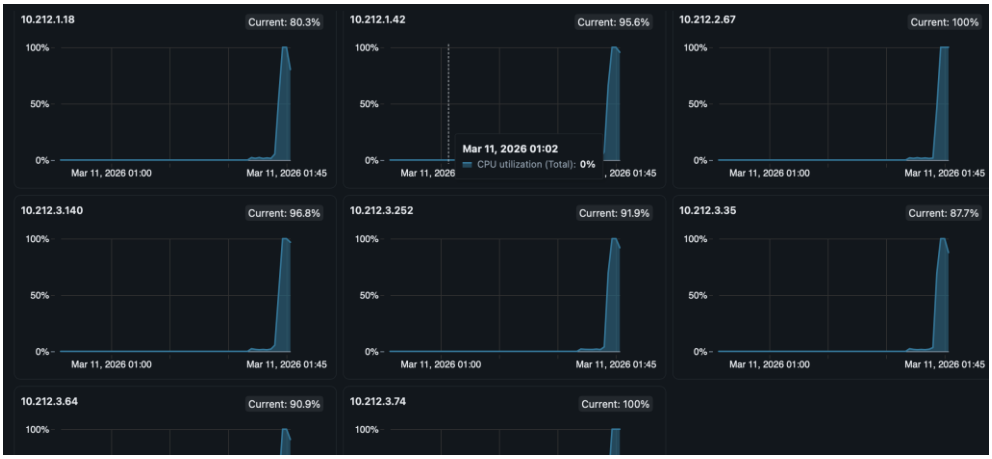

Fig 3: CPU Utilization by node when RAG pipeline is executed

Fig 4: CPU Utilization and Memory Utilization Of All Executor Nodes In Spark Only Cluster

3.7. Ray Cluster

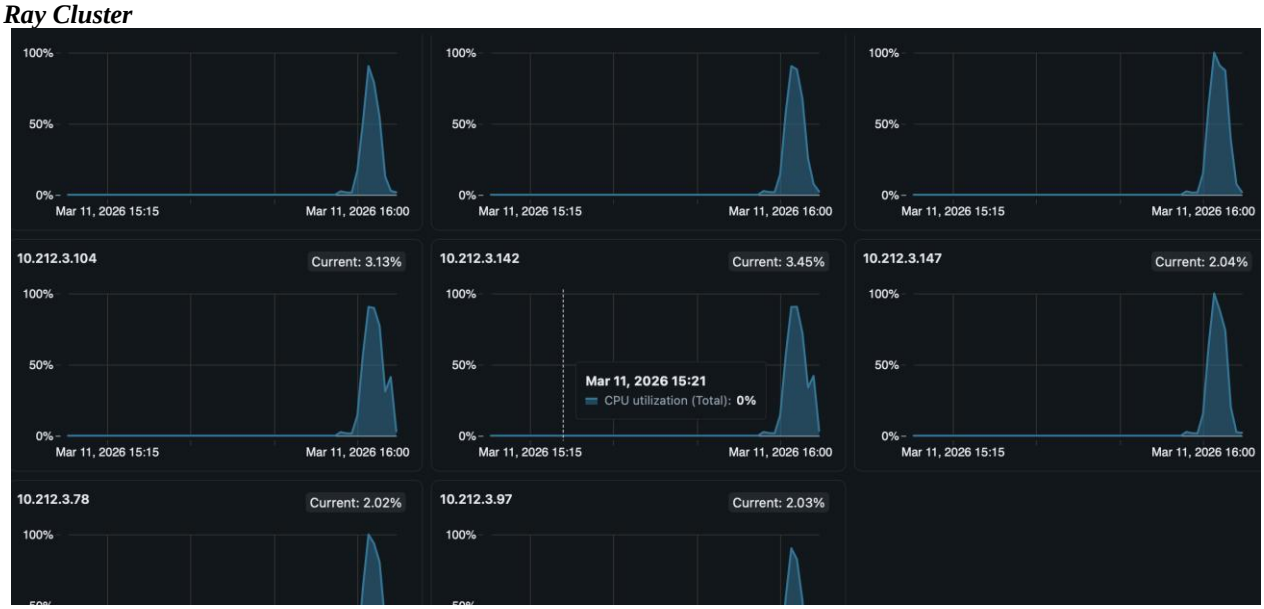

Fig 5: Individual Node CPU Utilization

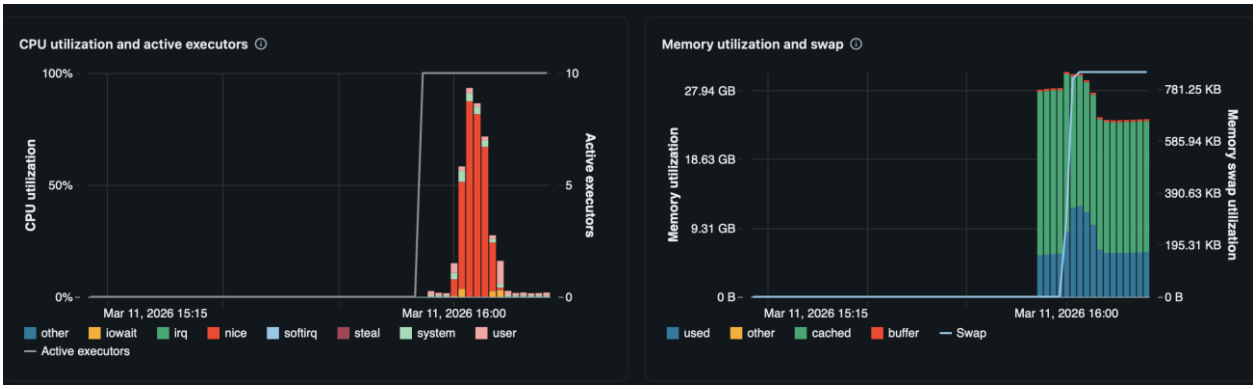

Fig 6: CPU Utilization and Memory utilization of all executor nodes on Ray cluster

4. Related Work: Comparison with Salesforce Agent force

Salesforce Agentforce represents a tightly integrated, platform-native solution for building autonomous agents within the Salesforce ecosystem. While effective for CRM-centric use cases, such proprietary solutions differ significantly from open, lakehouse-based architecture.

Salesforce Agentforce abstracts data ingestion, retrieval, orchestration, and execution within the Salesforce platform, enabling rapid deployment of CRM-focused agents. However, this abstraction limits architectural transparency, extensibility, and cross-platform integration. In contrast, the above proposed architecture adopts a decoupled, modular design that separates data ingestion, transformation, embedding generation, and semantic retrieval. This design aligns with emerging research on retrieval-augmented generation (RAG) systems that emphasize openness, explainability, and scalability.

Prior studies on enterprise RAG architecture highlight the importance of vector-based semantic retrieval, unified data governance, and independent scaling of compute and storage. The proposed architecture builds on these principles by leveraging Databricks Lakehouse for end-to-end data processing and a vector index for low-latency semantic search, while avoiding vendor lock-in inherent in proprietary agent platforms.

4.1. Cost Efficiency Quantitative Evaluation Criteria

With the proposed architecture above there will be some DBU's spent to load knowledge articles into Lakehouse initially. However, content consumption significantly **outpaces** content creation within Salesforce Knowledge and hence there will be very few writes after initial load. The Vector database like Pinecone(serverless) charges approximately $16 for 1 million read units per month and $0.33 per GB storage. Whereas Salesforce Agentforce charges $2 per conversation in a 24-hour period[5]. The proposed architecture demonstrates lower marginal cost at scale compared to Salesforce Agentforce.

5. Conclusion

This paper presented a scalable and extensible enterprise architecture that integrates Salesforce Knowledge, Databricks Lakehouse, and a vector index–based semantic retrieval layer to enable accurate and governed question answering and retrieval-augmented intelligence. By decoupling data ingestion, transformation, embedding generation, and retrieval, the proposed architecture addresses key limitations of tightly integrated, proprietary AI platforms while meeting enterprise requirements for scalability, transparency, and cost efficiency.

Through comparative analysis with Salesforce Agentforce, this study highlights the advantages of an open, lakehouse-centric design, particularly in terms of retrieval quality, architectural transparency, governance, and extensibility. The proposed approach enables independent scaling of compute and storage resources, flexible integration with orchestration frameworks such as LangChain, and support for advanced conversational and agentic AI workflows beyond CRM-bound use cases.

While the architecture offers clear benefits, it also introduces operational considerations related to system integration, index management, and embedding lifecycle maintenance. These trade-offs are offset by the increased control and reproducibility afforded by the modular design, making the solution well suited for both enterprise deployment and research experimentation.

Future work will focus on quantitative evaluation of retrieval and answer quality, optimization of incremental indexing strategies, and extension toward multi-agent orchestration and real-time knowledge updates. Additionally, incorporating feedback-driven learning and policy-aware governance mechanisms will further enhance the robustness and trustworthiness of the system.

References

1. Vector Search | Databricks (https://www.databricks.com/product/machine-learning/vector-search)
2. Vattam, L. (2022). Salesforce REST API in Action: A Practical and Research-Based Exploration of Integration Solutions. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(2), 36-43.
3. Gupta, S., Ranjan, R., & Singh, S. N. (2024). A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions. arXiv preprint arXiv:2410.12837.

4. Dive deep into vector data stores using Amazon Bedrock Knowledge Bases | Artificial Intelligence
5. Salesforce Agentforce Pricing | Salesforce
6. What is a Vector Database & How Does it Work? Use Cases + Examples | Pinecone
7. Sarmah, B., Mehta, D., Hall, B., Rao, R., Patel, S., & Pasquali, S. (2024, November). Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In Proceedings of the 5th ACM International Conference on AI in Finance (pp. 608-616).