Automated Knowledge Bank Construction for Business Intelligence LLMs

Joseph Standerfer
Amazon Web Services
Austin, TX, USA
joesta@amazon.com

Elisabeth Munger
Slalom
Seattle, WA, USA
liz.berg@slalom.com

Shayaan Naik
Amazon Web Services
New York, NY, USA
nashayaa@amazon.com

Abstract

This paper presents a novel approach to building automated knowledge banks for Generative Business Intelligence (GenBI) systems, enabling natural language interfaces to organizational data without specialized engineering expertise. We demonstrate how dashboard definitions can be transformed into knowledge repositories that bridge the semantic gap between Large Language Models (LLMs) and organization-specific data contexts. Our methodology extracts SQL from dashboards, generates AI-powered data dictionaries, and indexes business terminology to teach LLMs "your SQL, not just SQL." Implemented for AWS Marketing, this system leverages dashboards as "proven recipes" containing both technical implementation and business context, ensuring alignment without manual documentation. By treating dashboards as crystalized business intelligence—representing validated queries enriched with business terminology—we demonstrate a scalable GenBI approach that maintains nuanced understanding of metrics calculations and business definitions. Validation achieved 94% dashboard-to-SQL extraction success across all dashboard components, while evaluation showed 83% accuracy on unseen questions, rising to 97% for metrics directly visualized in source dashboards—demonstrating how automated knowledge extraction effectively powers natural language analytics while maintaining business context integrity.

CCS Concepts

• **Information systems** → **Business intelligence**; • **Computing methodologies** → *Natural language processing*; *Information extraction.*

Keywords

Generative AI, Business Intelligence, Retrieval-Augmented Generation, Data Discovery, Natural Language Processing

ACM Reference Format:
Joseph Standerfer, Elisabeth Munger, and Shayaan Naik. 2025. Automated Knowledge Bank Construction for Business Intelligence LLMs. In *Proceedings of KDD 2025 Workshop on Structured Knowledge for Large Language Models (KDD 2025 Workshop).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

KDD 2025 Workshop, Toronto, Canada

ACM ISBN 978-1-4503-XXXX-X/25/08
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Enterprise organizations face a fundamental challenge in deploying agentic AI systems: while Large Language Models excel at general reasoning and language understanding, they struggle with organization-specific contexts, terminology, and established business practices. This gap is particularly acute in business intelligence applications where subtle differences in metric calculations can have significant financial implications.

Traditional approaches to this problem, such as manual documentation, custom model training, or extensive prompt engineering, prove costly and difficult to maintain. Meanwhile, enterprises possess rich repositories of validated business logic in their existing dashboards, representing years of refined domain knowledge and approved calculation methodologies. These dashboards serve as "crystalized intelligence" but remain underutilized as knowledge sources for AI systems.

We present an enterprise agentic AI platform that bridges this gap through automated knowledge extraction from existing business intelligence assets. Our tool-augmented agent architecture transforms dashboard definitions into structured knowledge repositories, enabling natural language interfaces that understand both SQL syntax and organization-specific semantics. The system addresses key enterprise requirements:

- **No-code deployment**: Automatic extraction eliminates manual documentation
- **Tool-augmented reasoning**: Specialized tools for knowledge retrieval, SQL generation, and visualization
- **Enterprise integration**: Seamless connection to existing data platforms and BI systems
- **Explainability**: Full trace visibility into tool usage and knowledge sources
- **Trustworthiness**: Alignment with validated business logic from approved dashboards

Deployed for AWS Marketing teams analyzing lead flow process data, our system demonstrates how agentic AI can democratize data access while maintaining the rigor required for enterprise decision-making. By treating dashboards as comprehensive knowledge sources containing both technical implementation and business context, we enable rapid deployment of natural language analytics that understand "your SQL, not just SQL."

2 Related Work

Natural language interfaces to databases (NLIDB) have evolved significantly from early rule-based systems [4] to modern neural approaches. Recent text-to-SQL systems like PICARD [8] and DIN-SQL [7] demonstrate impressive performance on benchmarks but

often struggle with enterprise contexts where business terminology and metric definitions are critical [5].

Commercial BI platforms like Amazon QuickSight Q [1], Microsoft Power BI Q&A, and Tableau's Ask Data provide natural language capabilities but typically require significant manual setup to define topics and semantic layers. This limits their flexibility for ad-hoc exploration across domain boundaries.

Retrieval-augmented generation (RAG) has emerged as a promising approach for enhancing LLM performance in domain-specific tasks. Lewis et al. [3] introduced the RAG framework for knowledge-intensive NLP tasks. In parallel, systems like DBPal [11] have explored different approaches to database contexts by incorporating schema information and generating synthetic training data to improve NL-to-SQL translation models. However, creating effective knowledge bases for these systems remains labor-intensive, particularly in enterprise settings with complex data models and domain-specific terminology [10].

Knowledge extraction from existing BI assets represents a less explored direction. Setlur et al. [9] investigated how visualization context could improve natural language understanding in visual analysis systems, while Narechania et al. [6] proposed methods for generating analytic specifications from natural language queries for data visualization. Unlike these approaches that focus primarily on visualization artifacts, our work treats dashboards as comprehensive knowledge sources containing both technical implementation and business context, enabling automated extraction of SQL patterns, data structures, and business terminology without extensive manual curation.

3 System Architecture

The GenBI agent architecture consists of five interconnected components that transform natural language queries into data-driven insights, with particular emphasis on the knowledge discovery mechanisms:

(1) **User Interface Layer**: Multi-component interface featuring a chat interface for query submission, an image viewer for visualization display, and a response trace viewer for transparency into the agent's decision-making process.
(2) **Orchestration Layer**: GenBI Agent powered by Bedrock LLM with specialized prompts and orchestration patterns that guide knowledge retrieval operations and maintain conversation context.
(3) **Data Discovery Layer**: Central to our architecture, this layer contains a DataDiscovery router function coordinating between three specialized Lambda functions:

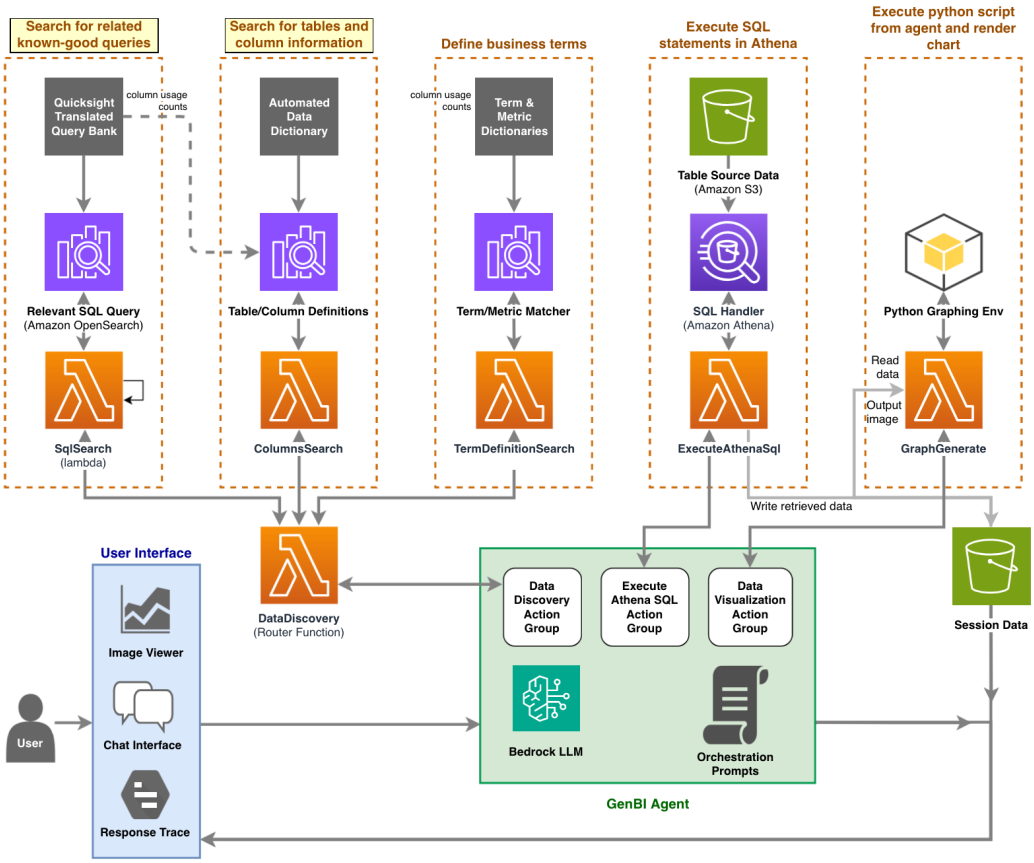

Figure 1: Generative BI Agent Implementation, Highlight Data Discovery Tools

- TermDefinitionSearch: Retrieves business terminology definitions
- ColumnsSearch: Identifies relevant table and column structures
- SqlSearch: Retrieves similar known-good SQL patterns

(4) **Execution Layer**: Components for SQL execution (ExecuteAthenaSQL) and visualization generation (GraphGenerate) that transform retrieved knowledge into actionable insights.

(5) **Data Storage Layer**: Underlying data sources and knowledge repositories, including:
- Table Source Data in Amazon S3
- Automatically extracted SQL knowledge bank indexed in OpenSearch
- Automated data dictionary with LLM-generated descriptions and comprehensive metadata
- Business terminology index for domain-specific concept alignment
- Session data store for maintaining context across interactions

The system follows a systematic workflow optimized for knowledge retrieval and context maintenance:

(1) User query is received through the chat interface
(2) The orchestration layer analyzes the query and determines required knowledge sources
(3) The DataDiscovery router coordinates parallel retrieval operations:
- Business terminology identification via TermDefinitionSearch
- Data structure discovery via ColumnsSearch
- Relevant SQL pattern retrieval via SqlSearch

(4) Retrieved knowledge is synthesized by the LLM to generate contextually appropriate SQL
(5) SQL is executed against Athena and results are transformed into visualizations
(6) Response and visualization are presented to the user while session context is maintained

This architecture ensures that the generated responses maintain business context alignment by leveraging the rich knowledge repositories that form the foundation of our approach. The emphasis on comprehensive knowledge retrieval before query generation is key to overcoming the semantic gap between general-purpose LLMs and organization-specific data contexts.

4 Implementation Details

Our implementation emphasizes automation throughout the knowledge extraction and organization process, transforming existing business intelligence assets into structured knowledge repositories without requiring specialized data engineering expertise. The Data Discovery Layer depends on three different Knowledge Base components (SQL, column metadata, and term definitions), each of which are described in this section. The SQL knowledge bank and column metadata knowledge bank comprise the heart of our solution: the automatic translation of dashboards and data tables into structured information indexed for LLM and agent reference.

4.1 SQL Knowledge Bank Implementation

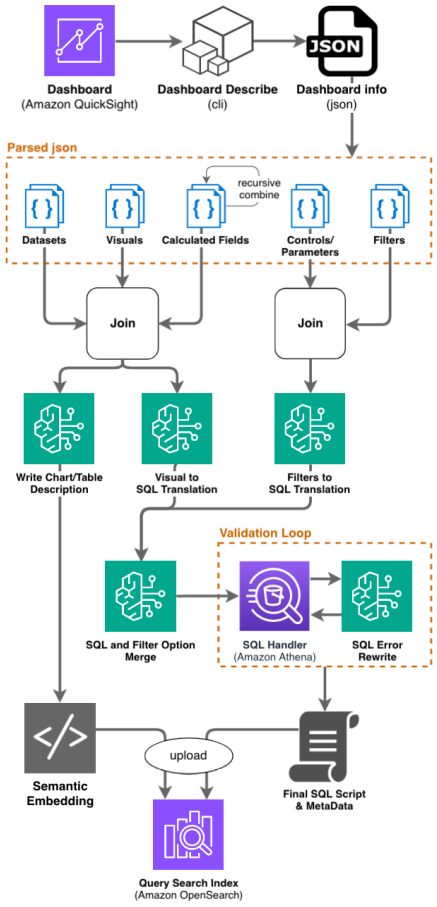

Figure 2: QuickSight Dashboard Query Extraction Process

A key innovation in our approach is the automatic extraction of SQL knowledge from existing QuickSight dashboards. These dashboards represent validated business logic that has been refined over time and serves as the organization's source of truth for key metrics. By transforming these dashboards into a structured SQL knowledge bank, we can teach LLMs organization-specific data practices without manual documentation. As illustrated in Figure 2, our methodology follows a multi-stage pipeline with parallel processing streams and feedback loops.

The architecture consists of five primary phases:

(1) **Dashboard Acquisition**: Using AWS CLI tools to retrieve complete dashboard definitions as JSON objects containing sheets, visuals, datasets, and control configurations.
(2) **Component Extraction**: Parsing of the dashboard JSON into five component types:
- Datasets: Source data references and schema information
- Visuals: Chart and table configurations with field mappings
- Calculated Fields: Custom business calculations with recursive dependencies

- Controls/Parameters: User-configurable filter parameters
- Filters: Pre-defined data constraints

(3) **Parallel LLM Processing**: Components are processed through distinct but complementary LLM workflows:
- *Left Stream*: Dataset, visual, and calculated field information is combined to generate both natural language descriptions and SQL translations
- *Right Stream*: Controls and filters are processed to generate SQL filter conditions
- These streams converge in a SQL and filter merging step to create complete queries

(4) **Validation Loop**: Generated SQL undergoes a rigorous validation process:
- Execution against actual data sources via Amazon Athena
- LLM-based error analysis and automated rewriting for failed queries
- Iterative refinement until success or maximum attempts reached

(5) **Dual Output Generation**: The process produces two complementary outputs that are stored in an OpenSearch index:
- Semantic embeddings of descriptions for vector search
- Validated SQL scripts with associated metadata

4.1.1 Recursive Calculation Resolution. Calculated fields are one source of complexity within the component extraction stage. Dashboard authors frequently create nested calculations that reference other calculated fields, forming a dependency graph that must be recursively resolved before SQL generation. Our system implements a recursive combination algorithm that:

(1) Identifies the complete set of calculated fields and their dependencies
(2) Constructs a directed dependency graph to determine evaluation order
(3) Recursively resolves expressions by substituting referenced calculations
(4) Converts dashboard-specific calculation syntax to standard SQL expressions

This approach enables accurate translation of complex business metrics that may involve multiple levels of calculation logic while preserving the original business intent.

4.1.2 LLM Choice. Our implementation employs a strategic combination of LLMs optimized for different tasks:

(1) **Initial Translation Tasks** (Visual Description, SQL Generation, Filter Translation):
- Model: Meta's Llama-3-70B-Instruct
- Rationale: Higher throughput, lower cost, and sufficient accuracy for initial translation

(2) **Refinement Tasks** (SQL Error Analysis, Query Rewriting):
- Model: Claude 3.5 Sonnet
- Rationale: Superior reasoning capabilities for complex error resolution and SQL debugging

This two-model approach balances efficiency and accuracy, reserving the more capable but costly model for complex reasoning tasks where its advanced capabilities provide the most value.

4.1.3 Case Study: Profit Segmentation Heatmap. To illustrate the methodology used to generate the SQL knowledge bank, we include an example of the extraction process for a complex HeatMap visual displaying profit segmentation by industry and customer segment from a publicly available business performance dashboard [2].

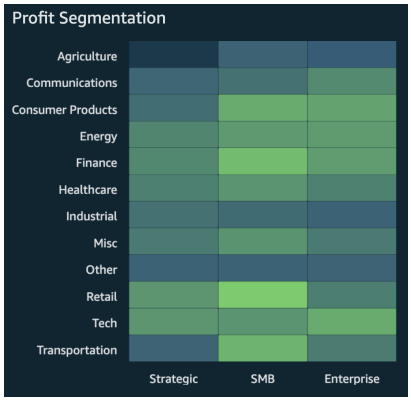

Figure 3: Profit Segmentation Heat Map from the AWS QuickSight Demo Central Business Summary Dashboard [2]

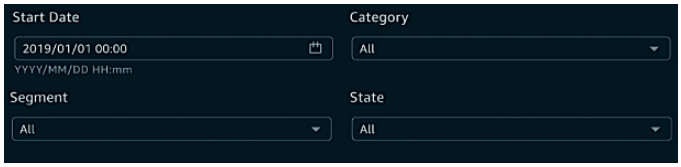

Figure 4: Dashboard control Bar [2]

(1) **Visual Definition Extraction**: The system extracted the complete visual definition, including field mappings (industry for rows, segment for columns, profit as value).
(2) **Initial Processing**: Two parallel streams processed the visual:
- Stream 1: Generated a business context description and base SQL query
- Stream 2: Identified applicable dashboard controls and converted them to SQL filter conditions

(3) **Merged Output**: The system combined the base query with filter conditions to create a complete SQL statement:

```
SELECT industry, segment, SUM(profit) AS profit
FROM dashboard_datasets.b2b_sales
WHERE date >= (SELECT MIN(date) FROM
 dashboard_datasets.b2b_sales)
AND segment IN (SELECT DISTINCT segment FROM
 dashboard_datasets.b2b_sales LIMIT 1)
-- Additional filters omitted for brevity
GROUP BY industry, segment
ORDER BY segment DESC

```

(4) **Validation Loop**: The system identified two issues during validation:

- First cycle: Syntax error with parameter placeholders
- Second cycle: Data type mismatch in the SUM function

(5) **Final Output**: After error correction, the system produced:
- A validated SQL query with proper type casting
- A semantic embedding of the visual description
- Associated metadata linking the query to its business context

4.1.4 Performance Analysis. The extraction process was evaluated on a business performance dashboard containing 47 visualizations across 14 different types, achieving:

- 100% success rate in SQL extraction (47 out of 47 visuals)
- 100% success rate in business context description generation
- 100% success rate in filter merging for applicable visuals
- 94% success rate in validation (44 out of 47 visuals)
- An average of 3.2 refinement attempts per visual

These results demonstrate the robustness of our approach across diverse visualization types and complex business logic patterns. The final knowledge bank created from these extractions was indexed in Amazon OpenSearch to support vector-based retrieval for the GenBI agent, enabling contextually accurate SQL generation that aligns with established business practices.

4.1.5 Amplifying Extraction Value through Control Combinations. While our system successfully extracted 47 SQL queries from dashboard visuals, the real power comes from how these queries can be combined with the various controls and filters extracted from the dashboard. Each visual typically supports multiple filter combinations (segment selections, date ranges, geographic filters), meaning a single extracted SQL query can serve as a template for thousands of potential variations. During the query retrieval process, our system identifies relevant SQL patterns from the original 47 extractions but can dynamically apply different filter combinations based on the user's natural language query. This combinatorial approach dramatically expands the effective coverage of our knowledge bank, enabling accurate responses to a much wider range of questions than the original dashboard visuals explicitly presented.

4.2 Data Dictionary Implementation

The data dictionary system employs a metadata extraction and enrichment pipeline, as illustrated in Figure 5. This pipeline transforms raw database schema information into searchable, semantically-rich column descriptions that bridge the gap between technical data structures and business terminology. This approach addresses common limitations of traditional data dictionaries, which often suffer from manual maintenance requirements, lack of business context, poor discoverability, limited search functionality, and scalability issues.

This automated data dictionary consists of four primary components:

(1) **Metadata Extraction Engine**: Utilizes AWS Athena to query database metadata and extract comprehensive statistics about tables and columns
(2) **Description Generator**: Leverages an LLM to create contextually-aware descriptions
(3) **Search Index**: Implements hybrid search through OpenSearch with both text-based and vector-based capabilities

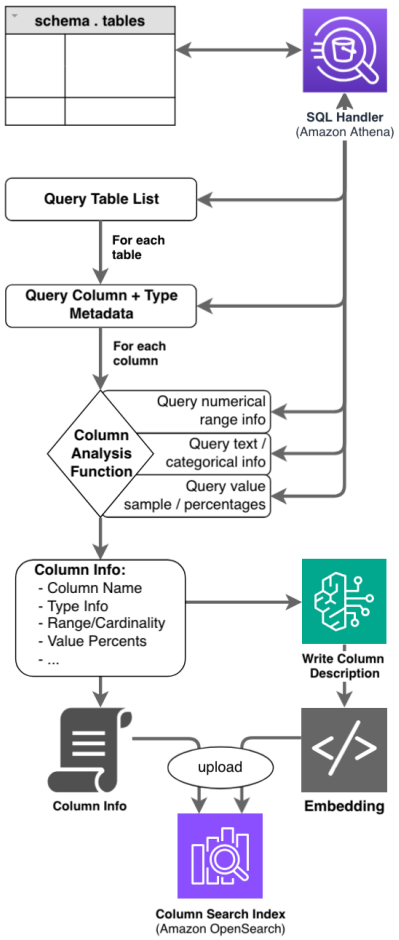

Figure 5: Automated Data Dictionary Generation Process

(4) **API Layer**: Provides programmatic access to the dictionary via Lambda functions

4.2.1 Metadata Extraction. Our implementation follows a multi-stage extraction process that progressively builds richer context for each database element:

(1) **Schema Exploration**: The process begins by querying the `schema.tables` metadata through Amazon Athena to identify all available tables in the target databases.
(2) **Table Iteration**: For each identified table, the system executes structured metadata queries to collect basic column attributes including names, data types, and constraint information.
(3) **Deep Column Analysis**: The core of our approach employs a specialized Column Analysis Function that executes three distinct types of analytical queries for each column:
- *Numerical Range Analysis*: For numeric columns, queries extracting distribution statistics including minimum/maximum values, averages, medians, and percentile breakdowns.

- *Text/Categorical Analysis*: For string and categorical columns, queries examining value patterns, format consistency, and distinct value counts.
- *Value Distribution Analysis*: For all column types, sampling representative values and calculating frequency percentages to identify dominant patterns.

These analyses are then transformed into structured metadata outputs containing column names, data types, range boundaries and distribution metrics, and foreign key relationships.

This progressive extraction approach ensures comprehensive metadata collection while managing query complexity and execution time through strategic parallelization and query optimization techniques. The extracted metadata serves two critical purposes: providing context for LLM-based description generation and enabling advanced search capabilities based on statistical patterns rather than just textual attributes.

4.2.2 Description Generation. A key innovation in our system is the application of Large Language Models to transform technical metadata into business-oriented column descriptions. Our approach:

(1) Constructs a comprehensive context vector for each column by combining its statistical signatures with table relationships and naming patterns
(2) Feeds this context through a specialized LLM prompt designed to emphasize business meaning over technical specifications
(3) Generates natural language descriptions that capture the column's purpose within its business domain context
(4) Incorporates value pattern insights to highlight typical usage patterns and constraints

This transformation bridges the gap between technical database structures and business terminology, creating descriptions that enhance discoverability without requiring detailed technical knowledge.

4.2.3 Hybrid Search Index. The system implements a dual-mode indexing strategy to support both keyword and semantic search capabilities:

(1) **Text-Based Indexing**: The complete column information, including the generated description, is indexed in Amazon OpenSearch with specialized analyzers optimized for database terminology.
(2) **Semantic Vector Indexing**: In parallel, the system generates embeddings of the column descriptions, creating dense vector representations that capture semantic meaning beyond exact terminology matches.

This dual-indexing approach enables flexible search strategies accommodating diverse user needs, from technical users seeking specific columns to business users exploring available data assets through natural language concepts.

Building on the dual-mode index, our search implementation combines multiple query strategies:

(1) **Custom Token Analysis**: Specialized text analyzers transform database naming conventions (snake_case, camelCase) into searchable tokens through custom tokenization rules.
(2) **Statistical Pattern Matching**: Search criteria can include value characteristics, allowing users to find columns with specific distribution patterns.
(3) **Vector Similarity Search**: Semantic queries leverage embedding-based similarity to identify conceptually relevant columns even when terminology differs.
(4) **Combined Ranking Algorithm**: Results from different search strategies are integrated through a weighted scoring system that balances exact matches with semantic relevance.

4.2.4 Comprehensive Search Strategies. Our system supports five complementary search strategies to accommodate diverse user needs:

(1) **Column Name Search**: For technical users familiar with naming conventions, including exact, phrase, and fuzzy matching.
(2) **Description Search**: For business users searching by concept, using semantic matching against AI-generated descriptions.
(3) **Hybrid Search**: Combining both approaches for comprehensive results, merging and deduplicating results from multiple search strategies.
(4) **Filtered Searches**: For narrowing results by technical criteria, including table-specific and data type filtering.
(5) **Natural Language Query Processing**: For conversational interfaces, extracting key search terms from natural language queries.

This multi-strategy approach enables effective data discovery for users with varying levels of technical knowledge and different search intents.

4.2.5 Usage & Optimization. To maintain efficiency across large data estates, the system implements several performance optimizations:

(1) **Parallel Query Execution**: Column analysis queries are executed in parallel batches, with dynamic sizing based on table complexity and system load.
(2) **Incremental Processing**: The system maintains execution state to process only new or modified tables during refresh cycles, minimizing redundant processing.
(3) **Optimized Query Design**: Statistical queries are designed with sampling and approximate aggregate functions when appropriate, balancing accuracy with performance.
(4) **Caching Layer**: Frequently accessed metadata and generated descriptions are cached to optimize response times for common queries.

Performance evaluation demonstrated the system's ability to process metadata for over 5,000 columns across 200+ tables in under 30 minutes, with search response times consistently below 200ms for typical user queries.

By integrating comprehensive metadata extraction, LLM-powered description generation, and sophisticated search capabilities, our data dictionary system enables intuitive data discovery that maintains the technical precision required for effective query generation

while accommodating the varied terminology needs of business users.

4.3 Business Terminology Index

To complement the technical metadata, we also implemented a business terminology index that captures domain-specific concepts and definitions. This component was created by scraping existing internal documents and organizing the terminology in an OpenSearch index for efficient retrieval. While less sophisticated than the other two discovery mechanisms, this component provides critical business context for interpreting user queries and aligning responses with organizational terminology.

4.4 GenBI Agent Implementation

The GenBI agent was implemented using Amazon Bedrock with structured knowledge retrieval capabilities and specialized action groups for data discovery and execution:

```
GenBI Agent (Orchestration Layer)
|-- Bedrock LLM
|-- Orchestration Prompts
`-- Action Groups
    |-- Data Discovery Actions
    |   |-- DataDiscovery Router Function
    |   |   |-- TermDefinitionSearch Lambda
    |   |   |-- ColumnsSearch Lambda
    |   |   `-- SqlSearch Lambda
    |-- Execute Athena SQL Actions
    |   `-- ExecuteAthenaSQL Lambda
    `-- Data Visualization Actions
        `-- GraphGenerate Lambda
```

The implementation leverages several AWS services in an integrated architecture:

- **Amazon Bedrock**: Provides foundational LLM capabilities with specialized knowledge-focused prompts
- **AWS Lambda**: Powers all functional components through serverless execution, with particular emphasis on the knowledge retrieval functions
- **Amazon OpenSearch**: Enables both vector and keyword search across all knowledge repositories, with specialized indices for SQL patterns, data definitions, and business terminology
- **Amazon Athena**: Serves as both a query execution engine and metadata source
- **Amazon S3**: Provides the underlying storage for both data and session state

The user interface consists of three complementary components:

- **Chat Interface**: Accepts natural language queries and displays textual responses with embedded references to discovered knowledge sources
- **Image Viewer**: Renders visualizations generated from query results
- **Response Trace**: Provides visibility into which knowledge sources were utilized during query processing, enhancing user trust and allowing for feedback on knowledge quality

Central to our implementation is the DataDiscovery Router Function, which coordinates knowledge retrieval operations across multiple specialized sources. This router ensures that each query leverages all relevant knowledge artifacts while maintaining session context between interactions.

The agent follows a knowledge-centric workflow for processing queries:

(1) Define relevant business terms using the terminology index, establishing domain context
(2) Identify appropriate tables and columns using the comprehensive data dictionary
(3) Retrieve similar known-good queries from the SQL knowledge bank extracted from dashboards
(4) Generate contextually appropriate SQL by synthesizing the retrieved knowledge
(5) Execute SQL against Athena and create visualizations using Python
(6) Store session context to maintain continuity across related queries

This approach enables natural language-driven data exploration that maintains business context alignment through specialized knowledge sources rather than requiring extensive model fine-tuning or manual prompt engineering for each query pattern.

5 Results

The GenBI system successfully achieved all five primary functional requirements identified for the proof of concept:

(1) Enable GenBI without BI/SQL familiarity
(2) Generate narrative and visual query outputs
(3) Minimize user setup costs
(4) Enable scalability across the organization
(5) Provide orchestrated agents with enhanced context

Our evaluation focused on two key dimensions: the system's ability to extract knowledge from dashboards and its performance in answering natural language queries using this extracted knowledge.

5.1 Dashboard Knowledge Extraction Performance

The system demonstrated robust performance in automatically extracting SQL knowledge from existing QuickSight dashboards:

- 94% dashboard-to-SQL extraction success rate across the full range of dashboard visuals and components
- 100% success rate in business context description generation
- Successful processing of 14 different visualization types with varying complexity

This high extraction success rate validates our approach of treating dashboards as comprehensive knowledge sources containing both technical implementation and business context.

5.2 Natural Language Query Performance

Using the automatically extracted and indexed knowledge, we evaluated the system's ability to answer unseen test questions across various metric categories:

The performance pattern strongly validates our central thesis: the system achieved 97% accuracy for queries about metrics directly

Table 1: Query Accuracy by Knowledge Alignment

Query Category	Accuracy
All unseen test questions (overall)	83%
Metric topics directly visualized in source dashboards	97%
Metrics requiring novel combinations or calculations	76%

visualized in source dashboards, demonstrating that dashboard-extracted knowledge serves as an effective foundation for natural language analytics. The overall 83% immediate accuracy on unseen test questions is particularly notable given that it relies solely on automatically extracted knowledge rather than manually curated SQL or descriptions.

These results demonstrate how automated knowledge extraction from existing assets can effectively power natural language analytics while maintaining business context integrity. The performance differential between visualized metrics (97%) and novel combinations (76%) also provides clear direction for improvement: expanding dashboard coverage or selectively augmenting the knowledge base for critical metrics not currently visualized.

6 Conclusion

This paper presented a scalable approach to implementing generative BI systems that leverage automated knowledge base construction from existing business intelligence assets. By treating dashboards as "proven recipes" containing both technical implementation and business context, we demonstrated how organizations can rapidly deploy natural language interfaces to complex business data without extensive manual documentation, data engineering expertise, or model fine-tuning.

The key innovations—automated SQL extraction from dashboards, AI-powered data dictionary generation, and business terminology indexing—provide a comprehensive foundation for LLM-based query processing that maintains business context alignment. This approach significantly reduces the implementation burden for generative BI systems while improving their accuracy and utility in domain-specific applications.

References

[1] Amazon Web Services. 2022. Amazon QuickSight Q—ML-powered BI question answering. Amazon Web Services, Inc.

[2] AWS QuickSight. 2024. *QuickSight Demo Central: Executive Business Summary Dashboard.* https://democentral.learnquicksight.online/#Analysis-DashboardDemo-Exec-Business-Summary

[3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.

[4] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84. doi:10.14778/2735461.2735468

[5] Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, Tianjiao Zhao, Amit Panalkar, Dhagash Mehta, Stefano Pasquali, Wei Cheng, Haoyu Wang, Yanchi Liu, Zhengzhang Chen, Haifeng Chen, Chris White, Quanquan Gu, Jian Pei, Carl Yang, and Liang Zhao. 2023. Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey. *arXiv preprint arXiv:2305.18703* (2023). doi:10.48550/arXiv.2305.18703

[6] Arpit Narechania, Arjun Srinivasan, and John Stasko. 2021. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 369–379. doi:10.1109/TVCG.2020.3030378

[7] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *arXiv preprint arXiv:2304.11015* (2023). doi:10.48550/arXiv.2304.11015

[8] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 9895–9901.

[9] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 365–377. doi:10.1145/2984511.2984588

[10] S. M Towhidul Islam Tonmoy, S M Mehedi Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. 2024. A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models. arXiv:2401.01313 [cs.CL]

[11] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, 2347–2361. doi:10.1145/3318464.3380589

A SQL Extraction Prompt Templates

These specialized prompts guide LLMs through different stages of the SQL extraction process, ensuring consistent and high-quality results.

A.1 SQL Extraction Prompt

```
You are an expert in analyzing Amazon Quicksight dashboard visuals.
      Your task is to interpret the visual definition and create
      corresponding SQL code.

For each visual, you will generate an SQL query that could be used
      to fetch the data for this visual.

Instructions:
- Create a single ANSI SQL statement (AWS Athena database) to supply
       the visual.
- Use the Source Table name(s) as the name of the source table(s).
- Focus ONLY on the actual data fields and calculations specified in
       the JSON structure.
- Use the Expression (if given) as the column calculation logic, and
       the ColumnName as the output column name in the SQL statement
      (Ex: Expression AS ColumnName).
- There may be a Name field in the JSON structure. Do NOT use Name
      as a SQL column name, always use the ColumnName as the SQL
      column name.
- If the visual is just an image file, or does not require tabular
      data, return an empty sql code block (Ex: ```sql\n```)
- Wrap your response in an sql code block so that it can be parsed.
      (Ex: ```sql\n(sql code...)```)
- Do not include any explanations or prose.
```

A.2 Description Generation Prompt

```
You are an expert in analyzing Amazon Quicksight dashboard visuals.
      Your task is to interpret the visual definition and generate a
      description.

For each visual, you will provide a brief description of what the
      visual represents.

Instructions:
- Provide one paragraph describing what information the chart is
      providing from a non-technical user's perspective
- Wrap your response in XML tags: <description>your description here
      </description>
- If the visual is just an image file, or does not require tabular
      data, return <description>N/A</description>
```

A.3 Filter Extraction and SQL Refinement

The system also employs specialized prompts for filter extraction and SQL refinement. The filter extraction prompt identifies dashboard controls and parameters, while the SQL refinement prompt handles error correction using detailed Athena error messages. Due to space constraints, these additional prompts are available in our online repository.

B Dashboard-to-SQL Case Study

To illustrate our knowledge extraction methodology, we present a complete case study of extracting SQL from a HeatMap visualization in a business performance dashboard.

B.1 Visual Identification and Processing

The process begins with identifying the "Profit Segmentation" HeatMap visual from the Business Performance Analysis dashboard. This visual displays profit distribution across industry segments and is part of a dashboard with multiple filtering controls.

The initial JSON definition includes configuration for rows (industry), columns (segment), and values (sum of profit). Our system extracts this information to understand the visualization's structure:


```
{
  "VisualId": "3787bc03-63e6-45d1-bbf2-7baffa915d97",
  "Title": {
    "FormatText": {
      "PlainText": "Profit Segmentation"
    }
  },
  "ChartConfiguration": {
    "FieldWells": {
      "HeatMapAggregatedFieldWells": {
        "Rows": [
          {
            "CategoricalDimensionField": {
              "Column": {
                "DataSetIdentifier": "B2B Sales",
                "ColumnName": "industry"
              }
            }
          }
        ],
        "Columns": [
          {
            "CategoricalDimensionField": {
              "Column": {
                "DataSetIdentifier": "B2B Sales",
                "ColumnName": "segment"
              }
            }
          }
        ],
        "Values": [
          {
            "NumericalMeasureField": {
              "Column": {
                "DataSetIdentifier": "B2B Sales",
                "ColumnName": "profit"
              },
              "AggregationFunction": "SUM"
            }
          }
        ]
      }
    },
    "SortConfiguration": {
      "HeatMapColumnSort": [
        {
          "Direction": "DESC"
        }
      ]
    }
  }
}
```


B.2 SQL Generation and Refinement Process

The system generates both a business description and SQL query from the visual definition. The initial SQL is:

```
SELECT industry, segment, SUM(profit) AS profit
FROM dashboard_datasets.b2b_sales
GROUP BY industry, segment
ORDER BY segment DESC
```

Next, the system identifies applicable dashboard filters and merges them into the SQL:

```
SELECT industry, segment, SUM(profit) AS profit
FROM dashboard_datasets.b2b_sales
```

```
WHERE date >= (SELECT MIN(date) FROM dashboard_datasets.b2b_sales)
AND segment IN (SELECT DISTINCT segment FROM dashboard_datasets.
      b2b_sales LIMIT 1)
AND category IN (SELECT DISTINCT category FROM dashboard_datasets.
      b2b_sales LIMIT 1)
AND ship_state IN (SELECT DISTINCT ship_state FROM
      dashboard_datasets.b2b_sales LIMIT 1)
GROUP BY industry, segment
ORDER BY segment DESC
```

During validation, the system identifies a data type issue with the profit column and automatically refines the SQL:

```
SELECT industry, segment, SUM(CAST(profit AS DOUBLE)) AS profit
FROM dashboard_datasets.b2b_sales
WHERE date >= (SELECT MIN(date) FROM dashboard_datasets.b2b_sales)
AND segment IN (SELECT DISTINCT segment FROM dashboard_datasets.
      b2b_sales LIMIT 1)
AND category IN (SELECT DISTINCT category FROM dashboard_datasets.
      b2b_sales LIMIT 1)
AND ship_state IN (SELECT DISTINCT ship_state FROM
      dashboard_datasets.b2b_sales LIMIT 1)
GROUP BY industry, segment
ORDER BY segment DESC
```

This refined SQL successfully executes against the database, completing the extraction process.

C Performance Metrics

C.1 SQL Extraction Success Rates

The dashboard SQL extraction process achieved the following results across the Business Performance Analysis dashboard:

- **Total Visuals Processed**: 47 across 5 sheets
- **SQL Extraction Success Rate**: 100% (47/47 visuals)
- **Description Extraction Success Rate**: 100% (47/47 visuals)
- **Filter Merging Success Rate**: 100% (37/37 applicable filters)
- **SQL Validation Success Rate**: 94% (44/47 visuals)
- **Average Refinement Attempts**: 3.2 attempts per visual

C.2 Success by Visualization Type

The system successfully handled 14 different visualization types with varying success rates:

Table 2: SQL Extraction Success Rate by Visual Type

Visualization Type	Count	Success Rate
Line Chart	26	96%
KPI	6	100%
Insight	2	50%
Bar Chart	2	100%
Pivot Table	2	50%
TreeMap	1	100%
Pie Chart	1	100%
Gauge Chart	1	100%
Heat Map	1	100%
Table	1	100%
Filled Map	1	100%
Box Plot	1	100%
Sankey Diagram	1	100%
Waterfall Chart	1	100%

The three visuals that didn't pass automatic validation contained complex date formatting logic, nested aggregation functions, or custom multi-part calculations. All required issues could be identified and fixed with minimal manual intervention.

D Core Implementation Components

This section highlights key implementation components of our extraction architecture.

D.1 QuicksightDashboardProcessor Class

The `QuicksightDashboardProcessor` class forms the backbone of our extraction system, handling the complex task of parsing dashboard definitions:

```
class QuicksightDashboardProcessor:
    def __init__(self, json_file_path: str, dataset_map: dict):
        self.json_file_path = json_file_path
        self.dataset_map = dataset_map
        self.dashboard_name = self._get_dashboard_name()
        self.dashboard_data = self._load_dashboard_data()
        self.calculated_fields_dict = self.
      _process_calculated_fields()

    def _process_calculated_fields(self) -> dict:
        """Process calculated fields from the dashboard definition
      """
        calculated_fields = self.dashboard_data.get('Definition', {})
      .get("CalculatedFields", [])
        calculated_fields = self._trace_expressions(
      calculated_fields)
        calculated_fields = self._process_dataset_identifiers(
      calculated_fields)
        return {(field['DataSetIdentifier'], field['Name']): field['
      Expression']
                for field in calculated_fields}

    def extract_visuals(self) -> pd.DataFrame:
        """Extract all visuals from the dashboard"""
        sheets = self.dashboard_data.get('Definition', {}).get('
      Sheets', [])
        visuals = []

        for sheet in sheets:
            sheet_id = sheet.get('SheetId')
            sheet_name = sheet.get('Name')
            sheet_visuals = sheet.get('Visuals', [])

            for visual in sheet_visuals:
                visual_data = {
                    'SheetId': sheet_id,
                    'SheetName': sheet_name,
                    'VisualId': visual.get('VisualId'),
                    'Title': self._extract_title(visual),
                    'Type': self._determine_visual_type(visual),
                    'Definition': visual
                }
                visuals.append(visual_data)

        return pd.DataFrame(visuals)
```

D.2 SQL Extraction and Refinement

Our system employs a two-model LLM strategy optimized for different extraction tasks:

```
class LLMExtractor:
    def __init__(self, llm: ChatBedrock):
        self.llm = llm

    def extract_sql(self, visual_definition: dict) -> str:
        """Extract SQL from a visual definition"""
        prompt = ChatPromptTemplate.from_messages([
```

```
            ("system", SQL_EXTRACTION_PROMPT),
            ("human", "Please use this Quicksight visual definition:\n{visual_def} to generate SQL.")
        ])

        chain = prompt | self.llm | StrOutputParser()
        response = chain.invoke({"visual_def": json.dumps(visual_definition)})

        # Extract SQL from code block
        sql = extract_code_block(response, "sql")
        return sql if sql else ""

    def refine_sql(self, original_sql: str, error_message: str) -> str:
        """Refine SQL based on error message"""
        prompt = ChatPromptTemplate.from_messages([
            ("system", SQL_REFINEMENT_PROMPT),
            ("human", "Original SQL:\n{sql}\n\nError:\n{error}")
        ])

        chain = prompt | self.llm | StrOutputParser()
        response = chain.invoke({"sql": original_sql, "error": error_message})

        # Extract refined SQL from code block
        sql = extract_code_block(response, "sql")
        return sql if sql else original_sql
```

D.3 Data Dictionary Implementation

The automated data dictionary system employs a sophisticated metadata extraction pipeline that progressively builds rich context for database elements:

```
class DataDictionaryExtractor:
    def __init__(self, aws_region: str, database_name: str, s3_output_dir: str):
        self.database = database_name
        self.aws_region = aws_region
        self.s3_output = s3_output_dir
        self.athena_client = AthenaClient(s3_output_dir)

    def get_column_value_info(self, table: str, column_name: str,
                              percent_threshold: float = 0.01) -> List[Dict[str, Any]]:
        """Get value distribution information for a specific column
        """
        percentage_query = f"""
        WITH total AS (
            SELECT COUNT(*) AS total_count
            FROM {self.database}.{table}
        ),
        value_counts AS (
            SELECT
                CASE WHEN {column_name} IS NULL THEN 'NULL'
                     ELSE CAST({column_name} AS VARCHAR)
                END AS val,
                COUNT(*) AS count
            FROM {self.database}.{table}
            GROUP BY
                CASE WHEN {column_name} IS NULL THEN 'NULL'
                     ELSE CAST({column_name} AS VARCHAR)
                END
        )
        SELECT
            val,
            count,
            CAST(count AS DOUBLE) / total.total_count AS percentage
        FROM value_counts, total
        WHERE CAST(count AS DOUBLE) / total.total_count > {percent_threshold}
        ORDER BY percentage DESC
        """

        result = self.run_query(percentage_query)
        return self._process_value_distribution(result)
```

E Knowledge Bank Structure

Each entry in the SQL knowledge bank contains rich metadata to facilitate effective retrieval:

```
{
  "id": "3787bc03-63e6-45d1-bbf2-7baffa915d97",
  "title": "Profit Segmentation",
  "visual_type": "HeatMapVisual",
  "sheet_name": "Summary",
  "dashboard_name": "Business Summary Dashboard",
  "description": "This heat map chart displays the total profit for
      different industries and customer segments...",
  "sql": "SELECT industry, segment, SUM(CAST(profit AS DOUBLE)) AS
      profit \nFROM dashboard_datasets.b2b_sales \nWHERE date >= (
      SELECT MIN(date) FROM dashboard_datasets.b2b_sales) \nAND
      segment IN (SELECT DISTINCT segment FROM dashboard_datasets.b2
      b_sales LIMIT 1) \nAND category IN (SELECT DISTINCT category
      FROM dashboard_datasets.b2b_sales LIMIT 1) \nAND ship_state IN
      (SELECT DISTINCT ship_state FROM dashboard_datasets.b2b_sales
      LIMIT 1) \nGROUP BY industry, segment \nORDER BY segment DESC",
  "source_tables": ["dashboard_datasets.b2b_sales"],
  "filters": [
    {
      "name": "Start Date",
      "type": "DateTimePicker",
      "column": "date",
      "description": "Filters data from the selected date forward"
    },
    {
      "name": "Segment Filter",
      "type": "Dropdown",
      "column": "segment",
      "multi_select": true
    }
  ],
  "business_context": {
    "purpose": "Strategic segmentation analysis for identifying high-
      value industry/segment combinations",
    "audience": "Sales leadership and account management teams",
    "usage_pattern": "Weekly business reviews, quarterly strategic
      planning",
    "related_metrics": ["Customer Profitability", "Industry Revenue
      Trend", "Segment Growth Rate"]
  },
  "technical_metadata": {
    "refinement_attempts": 2,
    "validation_success": true,
    "extraction_date": "2024-04-29"
  }
}
```

This structured format facilitates both semantic search through the description field and pattern matching through the SQL field, enabling effective retrieval of relevant examples during query generation.

F Search Implementation Details

The system implements a sophisticated hybrid search approach combining keyword-based and vector-based techniques:

F.1 Custom Token Analysis for Database Naming

```
"settings": {
    "analysis": {
        "analyzer": {
            "column_analyzer": {
                "type": "custom",
                "tokenizer": "standard",
```

```
                "filter": [
                    "lowercase",
                    "word_delimiter_graph",
                    "unique"
                ],
                "char_filter": [
                    "underscore_to_space"
                ]
            }
        },
        "char_filter": {
            "underscore_to_space": {
                "type": "pattern_replace",
                "pattern": "_",
                "replacement": " "
            }
        }
    }
}
```

F.2 Multi-Strategy Search

The search implementation employs multiple query strategies with weighted scoring:

```
"bool": {
    "should": [
        // Exact match on the normalized column name
        {
            "term": {
                "column_name.exact": {
                    "value": search_term,
                    "boost": 10.0
                }
            }
        },
        // Phrase match (words in exact order)
        {
            "match_phrase": {
                "column_name": {
                    "query": search_term,
                    "boost": 5.0,
                    "slop": 1
                }
            }
        },
        // Token match (individual words in any order)
        {
            "match": {
                "column_name": {
                    "query": search_term_no_underscores,
                    "operator": "and",
                    "boost": 3.0
                }
            }
        },
        // Description match
        {
            "match": {
                "description": {
                    "query": search_term,
                    "boost": 1.0
                }
            }
        }
    ],
    "minimum_should_match": 1,
    "filter": []
}
```

F.3 Vector Search Integration

The system also supports vector-based semantic search using AWS Bedrock embeddings:

```
class CustomVectorSearch:
    def __init__(self, opensearch_url: str, index_name: str,
                 region: str = 'us-west-2',
                 embedding_model_id: str = 'amazon.titan-embed-text-v2:0'):
        self.opensearch_url = opensearch_url
        self.index_name = index_name
        self.bedrock_embeddings = self.get_embedder(
            embedding_model_id=embedding_model_id,
            region=region
        )

    def get_embedder(self, embedding_model_id: str, region: str):
        return BedrockEmbeddings(
            region_name=region,
            model_id=embedding_model_id
        )

    def add_vectors(self, texts: List[str], embedding_contents: List[str],
                    metadatas: List[Dict[str, Any]], batch_size: int = 10):
        embeddings = self.embedding_function.embed_documents(embedding_contents)

        for i in range(0, len(texts), batch_size):
            texts_slice = texts[i:(i + batch_size)]
            embeddings_slice = embeddings[i:(i + batch_size)]
            metadatas_slice = metadatas[i:(i + batch_size)]

            self.add_embeddings(
                text_embeddings=[(t, e) for t, e in zip(texts_slice, embeddings_slice)],
                metadatas=metadatas_slice
            )
```

This hybrid approach enables both precise technical matching for users familiar with database terminology and more exploratory semantic search for business users approaching the system with natural language questions.