# Role of Databases in GenAI Applications

**Santosh Bhupathi**

Sr. Solutions Architect
AWS

**Abstract:**
**Generative AI (GenAI) is transforming industries by enabling intelligent content generation, automation, and decision-making. However, the effectiveness of GenAI applications depends significantly on efficient data storage, retrieval, and contextual augmentation. This paper explores the critical role of databases in GenAI workflows, emphasizing the importance of choosing the right database architecture to optimize performance, accuracy, and scalability. It categorizes database roles into conversational context (key-value/document databases), situational context (relational databases/data lakehouses), and semantic context (vector databases) each serving a distinct function in enriching AI-generated responses. Additionally, the paper highlights real-time query processing, vector search for semantic retrieval, and the impact of database selection on model efficiency and scalability. By leveraging a multi-database approach, GenAI applications can achieve more context-aware, personalized, and high-performing AI-driven solutions.**

**Keywords: Database, GenAI, AI, Cloud technologies, Vector Database.**

## Introduction (Generative AI & Large Language Models)

Generative AI (GenAI) represents a transformative leap in artificial intelligence, leveraging advanced models such as Transformers, GPT-4, and Gemini to generate human-like content across multiple modalities [1], [2]. Unlike traditional AI models that focus on classification or predictive tasks using predefined patterns, GenAI utilizes deep learning architectures like Transformer-based Large Language Models (LLMs) [2] to create text, images, code, and audio.

Prominent GenAI models include GPT-4 for advanced text generation [1] and Google's Gemini for multimodal AI applications [2]. These models leverage massive datasets and training methodologies such as Reinforcement Learning with Human Feedback (RLHF) [3] and retrieval-augmented generation (RAG) [4] to enhance their contextual understanding and adaptability.

These AI models, trained on large-scale data, can understand context, generate creative outputs, automate workflows, and drive innovation across industries. GenAI is transforming fields such as healthcare (AI-assisted diagnosis and drug discovery [5]), finance (automated risk analysis and fraud detection [6]), customer support (intelligent virtual assistants [7]), and software development (AI-driven code generation [8]). The emergence of multimodal AI which enables models to process and generate text, images, and audio simultaneously is further unlocking new possibilities in automation, personalization, and decision-making.

## Value Proposition of Generative AI Applications

Generative AI applications deliver substantial business value by enhancing efficiency, creativity, and decision-making. Below are some key value propositions:

**1. Enhanced Productivity & Automation**

- Automates repetitive tasks like document generation, summarization, and code completion, reducing manual effort and improving efficiency.
- Enables self-service customer support with AI-powered chatbots that provide contextual and natural interactions.

**2. Personalized User Experience**

- Powers hyper-personalization by generating tailored content, recommendations, and responses based on user preferences and context.

- Enhances marketing efforts with AI-generated ad copies, email campaigns, and personalized product recommendations.

**3. Intelligent Decision-Making**

- Helps in real-time data analysis and insights extraction for better decision-making in finance, healthcare, and operations.
- Improves fraud detection, risk analysis, and compliance monitoring by analyzing vast amounts of structured and unstructured data.

**4. Creativity & Content Generation**

- Generates high-quality content, including articles, product descriptions, scripts, and creative writing for media and marketing.
- Assists in design, music, and video generation, reducing creative bottlenecks and accelerating production cycles.

**5. Cost Optimization & Scalability**

- Reduces costs associated with manual content creation, customer support, and software development by automating key processes.
- Scales seamlessly across global operations, enabling faster market expansion and localized content generation.

**6. Democratization of AI & Innovation**

- Enables non-technical users to leverage AI-driven tools for content creation, analysis, and automation.
- Empowers developers, researchers, and enterprises to build new AI-driven applications without extensive AI expertise.

**From Bits to Brains: Why the Right Database Matters in GenAI**

In many GenAI applications, there's a common misconception that only a vector database is needed, particularly for semantic search and embedding management [9], [10]. However, different parts of the application often require specialized storage solutions. Vector databases undoubtedly play a pivotal role in semantic context, but relying on them exclusively can lead to performance bottlenecks, data inconsistencies, and an incomplete view of user interactions [11]. By matching each data type to the most appropriate database technology, GenAI solutions can deliver faster, more accurate, and contextually rich responses, ensuring the true potential of AI-driven applications is fully realized [12].

In GenAI applications, real-time data processing demands databases that can handle rapid queries and large-scale ingestion without bottlenecks [13]. Traditional database systems often struggle with the high throughput required to train and deploy AI models, making scalability and low latency critical. Continuous model updates also mean that databases must support efficient retrieval mechanisms to ensure model integrity. Ultimately, selecting the wrong database can lead to performance degradation, compromised accuracy, and diminished overall effectiveness of GenAI solutions.

**Role of Databases in Generative AI Applications**

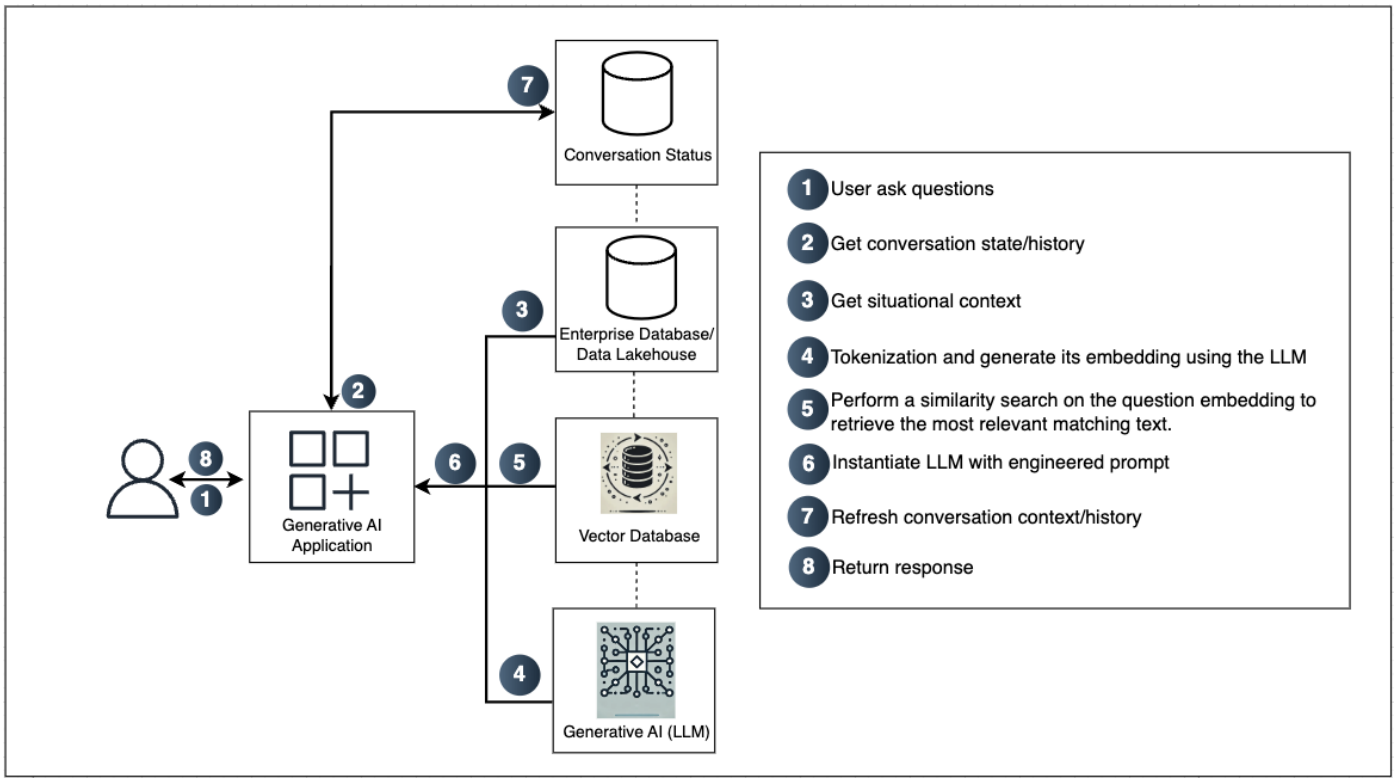
Databases play a critical role in Generative AI (GenAI) applications, ensuring efficient data storage, retrieval, and contextual augmentation for AI-driven responses [14], [15]. When a user interacts with a GenAI-powered system, the application relies on multiple databases to fetch historical context, enrich responses with enterprise knowledge, and enhance overall accuracy.

In the user's critical path, databases contribute to key stages of the workflow, enabling AI models to make informed decisions by leveraging structured and unstructured data. When an end-user interacts with a Generative AI application, they typically post a question or prompt to the system. At first glance, this may seem like a simple request-response mechanism, but behind the scenes, a complex orchestration of databases and AI models takes place to deliver accurate, context-aware, and relevant responses [15].

In this blog, we'll explore how databases play a crucial role in enhancing Generative AI applications by providing different types of contextual data that shape the final response. Below is an overview of how databases are used at different steps of a GenAI interaction.

**How Databases Support the User's Critical Path in GenAI Applications**

Here is a high-level workflow of a GenAI Application.




**Step 1: User Asks Questions**

The user initiates an interaction by submitting a query, request, or instruction to the GenAI application. This can range from simple questions to more complex tasks requiring in-depth analysis or content generation. The system captures this input and sets the stage for subsequent steps, ensuring it has a clear starting point for generating a relevant, context-aware response.

**Database Role:**

- At this point, no direct database interaction is strictly required however, the question itself will later be stored or logged for future reference and to maintain conversation continuity. Typically, the application logs the user's query in a database for potential auditing, analytics, or future reference. If user or session data is required at this point such as user authentication or preferences then the system may query a relational or key-value store to validate or personalize the request before moving to the next steps.

**Step 2: Understanding User Interaction: Query Processing & Contextual Data Retrieval**

The user submits a query to the Generative AI application, as soon as the user asks a question, the application loads a relevant prompt template. This template engineering process enhances the original question by adding additional context, ensuring the Large Language Model (LLM) produces a more accurate and relevant output. **Conversational Context (Maintaining Chat History)** refers to the process of preserving and managing all previous interactions between the user and the system. This includes user queries, the system's responses, and any relevant metadata or context. By storing and referencing this historical data in an enterprise database or data Lakehouse application can provide more coherent and personalized responses in subsequent interactions. Essentially, it enables the AI to "remember" what was previously discussed, thereby improving continuity and user experience in a conversational setting.

**Database Role:**

- Since LLMs do not retain memory, the system stores previous user interactions in a database.
- This ensures continuity in conversations, making the AI responses more coherent and contextually relevant over multiple interactions.
- Enterprise Database / Data Lakehouse: Stores long-term conversation data or user session logs. When the application needs the historical context such as the user's previous questions, system instructions, or any relevant session metadata it queries this database.

- Why It Matters: Persisting conversation data ensures the application can reference past interactions, enabling more coherent, context-aware responses.

**Step 3: Context Enrichment & Knowledge Retrieval**

Large Language Models (LLMs) are inherently stateless, treating each query as if it were their first. They lack any built-in mechanism to recall previous interactions or shared information, making an external data store indispensable for preserving context. Databases fill this need by retaining conversation history, user profiles, and supplementary knowledge ultimately enriching the AI's ability to generate coherent, context-aware responses.

**Situational Context (User Profile & Operational Data)**

This involves gathering any additional information the system may need to provide a well-informed response. Examples include user profile details (such as preferences or history), operational data (like current system status or relevant business metrics), and any domain-specific knowledge. By querying an enterprise database or data Lakehouse at this stage, the application retrieves the contextual data needed to tailor responses accurately and ensure relevance.

**Database Role:**

- The application also needs user-specific and real-time data to generate a personalized response.
- If the system requires sub-millisecond latency, A caching service is used for fast retrieval of frequently accessed data.
- The system queries a conversation history database to retrieve past interactions and user preferences.
- If the AI application supports personalization, relational databases may fetch user-specific data to tailor responses.

**Step 4&5: LLM-Based Embeddings and Vector Search: From Tokenization to Semantic Retrieval**

The application tokenizes the original question, converting the text into a numerical form known as embeddings. These embeddings, generated by the large language model (LLM), capture the semantic meaning of the user's query. The embeddings serve as a representation of the question in a way that helps the system understand the underlying context, enabling more accurate and contextually relevant responses. The process of embedding generation using the LLM facilitates the transformation of raw text into meaningful, machine-readable representations essential for downstream tasks.

**Semantic Context: Tokenization and Embedding Generation with the LLM**

The application begins by converting the user's query into a sequence of tokens effectively translating text into a numerical form. This process, known as embedding generation, leverages a large language model (LLM) to produce high-dimensional vector representations that capture semantic meaning. By creating these embeddings, the LLM provides a nuanced context for the user's question, laying the groundwork for more accurate and context-aware responses.

**Similarity search (Vectorized Knowledge Retrieval)**

Perform a Similarity Search on the Question Embedding in the GenAI workflow. Here, the system leverages a vector database to retrieve semantically similar documents or data points by comparing the numerical embeddings generated in the previous step. This allows the application to provide contextually rich and relevant information essentially capturing the "meaning" of the user's query rather than relying solely on keyword matches.

At this stage, all three types of contexts (conversational, situational, and semantic) are synthesized into an engineered prompt to provide the LLM with the best possible input. This enables semantic search, allowing the model to find relevant information even when the exact words differ.

**Database Role:**

- To enhance understanding, the AI converts the user's query into embeddings (mathematical representations of text). Need a specialized database called vector database.
- These embeddings are then searched against a Vector Database to retrieve similar text or knowledge snippets.
- Knowledge Graphs and vector databases assist in semantic search, allowing the model to find relevant facts and documents before generating a response.

- Vector databases perform similarity searches on pre-indexed text, images, or other embeddings to find relevant context for the AI model.
- This step is essential for RAG (Retrieval-Augmented Generation), where AI models ground their responses with real-time or proprietary knowledge instead of relying solely on pre-trained data.

**Understanding Vector Databases: The Engine Behind Semantic Search**

A vector database is a specialized data store designed to handle high-dimensional numerical vectors, often called embeddings. A vector database indexes and stores vector embeddings for fast retrieval and similarity search. In GenAI applications, these embeddings represent the semantic meaning of text, images, or other data types [16]. By storing and indexing these embeddings, vector databases enable efficient similarity searches based on conceptual proximity rather than exact string matches.

**Why Use a Vector Database for Semantic or Similarity Search?**

1. **High-Dimensional Data**: GenAI models frequently produce embeddings that can have hundreds or thousands of dimensions. Vector databases provide specialized indexes (e.g., IVF, HNSW) optimized for performing *approximate nearest neighbor* (ANN) searches at scale [18].
2. **Semantic Matching**: Instead of matching exact keywords, vector databases identify conceptually related results even if the words differ. This is essential in scenarios where synonyms or paraphrased text must be recognized as relevant.
3. **Low Latency at Scale**: Vector databases are built to handle large volumes of embeddings while still delivering fast query times. Traditional databases often struggle with performance when tasked with similarity searches in high-dimensional space.
4. **Integration with GenAI Pipelines**: By storing model-generated embeddings, vector databases streamline the retrieval step in many GenAI workflows, such as retrieving semantically related documents for question answering or recommendation systems.

**Why Not Use a Typical Relational or NoSQL Database?**

1. **Lack of Specialized Indexing**: Relational and many NoSQL databases are optimized for structured data lookups or key-based queries, not high-dimensional similarity searches. They do not typically support the specialized indexing structures needed for efficient ANN operations.
2. **Performance Bottlenecks**: Attempting to store and query large numbers of embeddings in a standard database can lead to high latency and resource usage, as these systems are not designed for vector-based lookups[18].
3. **Limited Query Capabilities**: While you can force a relational or NoSQL database to store embeddings, you'd still need an external library or application logic to perform similarity searches. This approach is often cumbersome, less efficient, and difficult to scale.
4. **Scaling Complexities**: Vector databases are built with distributed architectures and optimized data structures specifically for scaling similarity search workloads. Adapting a relational or general-purpose NoSQL store for these tasks can become unwieldy or prohibitively expensive.

Let's explore this vector database with an example use-case of an e-commerce store selling running shoes, where we'll see how semantic search helps users find the perfect pair within their budget.

1. **The Problem: Keyword Search vs. Semantic Search**

Many e-commerce platforms rely on keyword-based search engines. While these systems can handle exact matches (e.g., "red running shoes"), they often miss nuanced queries like "comfortable sneakers under $100." This gap leads to missed opportunities and a poor user experience.

Why It Matters

- Enhanced User Satisfaction: Users find products that truly match their intent.
- Improved Conversions: More relevant results lead to higher purchase rates.

2. **Sample Inventory Data**

Consider a running shoe inventory. For simplicity, each product is represented by a 2D vector embedding (real embeddings often have hundreds of dimensions).

Here's our sample inventory in tabular form:

| ID | Product | Price | Category | Description | 2D Embeddings |
|---|---|---|---|---|---|
| 1 | Nike ZoomX Infinity Run | $120 | Running Shoes | Lightweight shoes with extra cushioning | [1.2, 3.5] |
| 2 | Adidas UltraBoost | $180 | Running Shoes | Premium shoes with high-end comfort | [2.0, 3.2] |
| 3 | Reebok Floatride | $90 | Running Shoes | Comfortable shoes for daily training | [3.1, 2.9] |
| 4 | ASICS Gel-Kayano | $110 | Running Shoes | Supportive design for stability and longer runs | [2.5, 3.0] |

**3. User Query**

A user visits the e-commerce site and types: "I need comfortable running shoes under $100."
Then A text embedding model (like Sentence-BERT) converts this query into a 2D embedding:

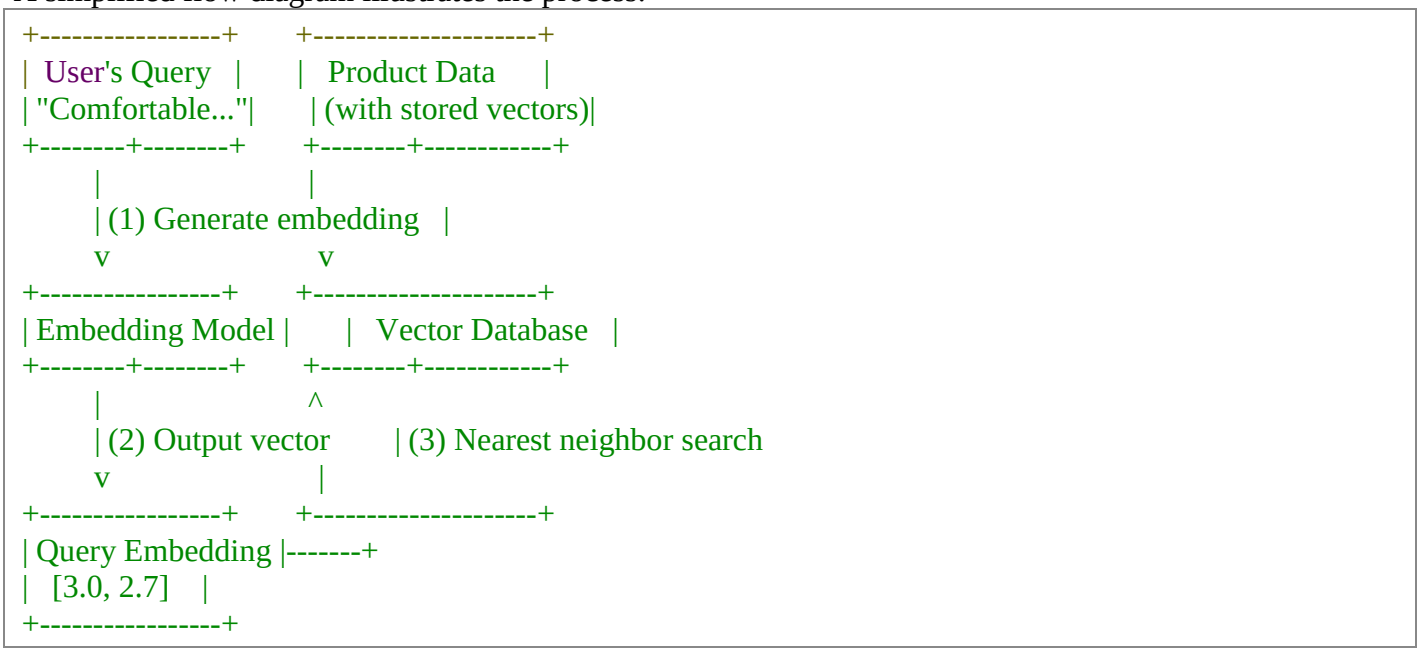
1. Query Embedding: [3.0, 2.7]
2. Note:(For production, embeddings typically have 128–1024 dimensions; we use 2D here for clarity.)

**4. Vector Database Search**

Instead of relying on keyword matching, we store the product embeddings in a **vector database** (such as Milvus, Pinecone, or Vespa). When the user submits the query:

1. The application **generates** the query embedding [3.0, 2.7].
2. The vector database performs a **nearest neighbor search** to find the most semantically similar products.
3. **Metadata filters** (e.g., "under $100") are applied to ensure only budget-friendly items are returned.

A simplified flow diagram illustrates the process:




**5. Distance Calculations (Euclidean)** [19]

To determine the similarity between the query and each product, we use Euclidean distance, which is the straight-line distance between two points:

$$distance(A, B) = \sqrt{(xA - xB)^2 + (yA - yB)^2}$$

Calculated distances from the query embedding [3.0, 2.7] to each product are as follows:

1. Nike ZoomX Infinity Run ([1.2, 3.5]):

$$\sqrt{(1.2 - 3.0)^2 + (3.5 - 2.7)^2} = 1.97$$

2. Adidas UltraBoost ([2.0, 3.2]):

$$\sqrt{(2.0-3.0)^2+(3.2-2.7)^2} = 1.12$$

3. Reebok Floatride ([3.1, 2.9]):

$$\sqrt{(3.1-3.0)^2+(2.9-2.7)^2} = 0.22$$

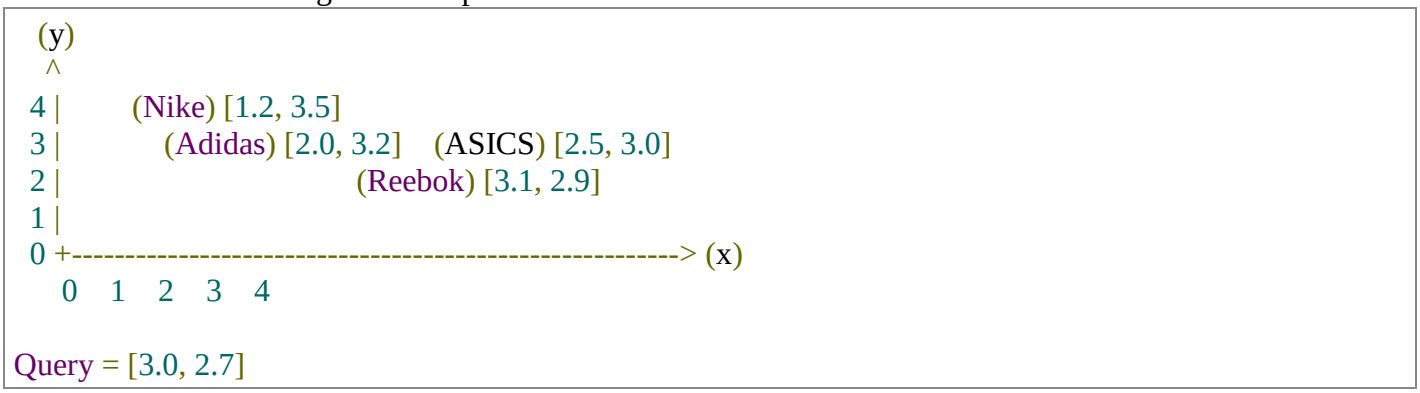
4. ASICS Gel-Kayano ([2.5, 3.0]):

$$\sqrt{(2.5-3.0)^2+(3.0-2.7)^2} = 0.58$$

Reebok Floatride is the closest match with a distance of 0.22, and it also meets the price criteria (under $100).

**6. Euclidean Distance Visualization**

Visualize the embeddings on a 2D plane:




**7. The Result**

After applying the similarity search and filtering by price:

- Top Recommendation: Reebok Floatride (Distance = 0.22, Price = $90)
- Runner-Up: ASICS Gel-Kayano (Distance = 0.58, Price = $110)
- Other products either fall short in semantic relevance or exceed the budget.

The vector database delivers these results quickly, ensuring that even subtle nuances in user intent are matched with the most appropriate product.

**Key Takeaways**

1. Semantic vs. Keyword: By converting text into embeddings, the system captures the true meaning behind queries and product descriptions, not just exact string matches.
2. Vector Databases: Designed to store and query high-dimensional vectors, they provide efficient and scalable semantic search capabilities.
3. Hybrid Queries: Combining semantic search with metadata filters (such as price) yields highly relevant and user-specific recommendations.
4. Visualization and Intuition: Using Euclidean distance in a 2D plot helps illustrate how semantic similarity is determined, making the concept more accessible.

Putting all these steps together, we start by recognizing the limitations of keyword-based searches, which often fail to capture the user's true intent. We then prepare our data by converting product descriptions into vector embeddings using a language model, ensuring each item's semantic attributes are preserved. These embeddings are stored in a vector database, specifically designed for efficient similarity searches on high-dimensional data. When a user submits a query, we generate an embedding for that query and perform a nearest neighbor lookup against our stored product vectors. Next, we refine the results by applying any necessary metadata filters such as price or brand and rank them based on their computed distance or similarity scores. Visualizing the data in a 2D plot (for demonstration purposes) helps illustrate how these distances translate into meaningful matches. Ultimately, the system delivers contextually relevant product recommendations that not only match the user's stated preferences but also account for subtle nuances, creating a more intuitive and effective search experience.

**Tip! Vector indexing for better performance**
To optimize similarity searches, leverage specialized indexing structures such as IVF, HNSW, or PQ-based approaches. These indexes enable the vector database to quickly narrow down candidate vectors, significantly reducing query times while maintaining high accuracy even at scale.

**Step 6: AI Model Execution & Prompt Engineering**
In this step, the application constructs a carefully engineered prompt by integrating the user's query, relevant context retrieved from prior interactions, and any auxiliary knowledge necessary to enhance the response. This comprehensive prompt is then passed to the large language model, which synthesizes the aggregated information to generate a contextually accurate and coherent answer. By leveraging both the user's question and the retrieved context, the LLM is able to provide a tailored, informative response that directly addresses the user's needs.
**Database Role:**
- No direct database query happens at this step, but previously retrieved data ensures that the AI model generates a well-informed response.
- In some applications, embeddings generated at this stage are also stored in a vector database, streamlining future similarity searches and ensuring the LLM has immediate access to contextually relevant information for subsequent prompts.

**Step 7: Updating Conversation State & Storing User Interactions**
After the large language model generates a response, the application must update its records to reflect the latest exchange. This involves capturing the user's question, the LLM's response, and any relevant metadata such as timestamps, user identifiers, or session IDs. By preserving these details, the system maintains a comprehensive conversation history, which can be used for future context retrieval, analytics, and model improvements.
**Database Role:**
- A Key-Value or Document logs the user query, AI-generated response, and metadata (timestamps, feedback, etc.).
- In enterprise settings, structured responses may be logged in a relational database for future audits and analytics.

**Step 8: Response Generation & Delivery**
Once the large language model (LLM) has produced its output, the system finalizes and delivers the response to the user. This stage can involve several sub-processes:
1. **Post-Processing and Formatting**:
   - The raw LLM output may be refined or formatted according to the application's requirements (e.g., ensuring consistency with UI guidelines or applying a content filter to remove sensitive data).
   - Additional metadata, such as response confidence scores or relevant references, can be appended for user clarity or internal logging.
2. **Delivery Mechanism**:
   - The response is then sent through the appropriate communication channel such as a web interface, mobile application, or API endpoint to reach the user.
   - In some cases, the system may adapt the presentation style based on the user's device or preferences (e.g., text-only versus rich media).
3. **Logging and Analytics**:
   - Simultaneously, the system may log details of the interaction like response time, content, or user feedback in a database for performance monitoring and future analysis.
   - These records enable developers to assess the LLM's effectiveness, detect issues, and iteratively refine prompts or model parameters.

By handling the final output with care, the application ensures users receive a coherent, context-aware answer while also capturing essential data for continuous improvement.

**Database Role:**

- If the system includes caching mechanisms, frequently used responses may be fetched from a low-latency database to optimize performance.
- The system may also use historical user interactions to refine responses over time via continuous learning and fine-tuning.

By understanding the unique characteristics of conversational, situational, and semantic contexts in GenAI applications helps ensure the right database is used for each type of data, optimizing performance and scalability. By selecting the most appropriate database for each context, applications can achieve faster query responses, more relevant results, and improved overall efficiency in managing diverse data types. This approach brings value by aligning database capabilities with the specific needs of the context, resulting in more accurate, personalized, and context-aware user experiences.

Comparing these context types clarifies their distinct data requirements and retrieval patterns, guiding more precise database selection. By aligning each context with a fitting storage solution, teams can achieve better performance, scalability, and reliability. Ultimately, this approach maximizes the quality and speed of GenAI responses, ensuring they are both contextually rich and efficient.

Below is a comparative breakdown of conversational context, situational context, and semantic context in GenAI applications, highlighting the most suitable database types, the reasoning behind these choices, and relevant open-source examples.

**Conversational Context**

Conversational context in GenAI applications involves storing chat history, user interactions, and exchanged messages to ensure coherent responses across sessions. This context is dynamic and unstructured, requiring rapid read/write operations for real-time updates.

**Database Type:** Key-Value / Document Databases (for flexibility and speed) or Relational Databases (if strong consistency is required).

Reason: Flexibility & Speed Schema-less storage supports varying conversation formats and enables fast updates and retrievals in real-time.

**Transaction Pattern:**

**High-Frequency Writes:** Continuous insertion of new messages as conversations evolve.

**Frequent Reads:** Quick retrieval of recent or historical chat logs to maintain context.

**High Concurrency:** Multiple simultaneous user sessions, each generating or reading messages in real time.

**Open-Source Examples:** Redis (key-value) [20]**,** MongoDB (document) [21]**,** PostgreSQL (relational) [22][23]

**Situational Context:**

Situational context consists of structured data like user profiles, operational metrics, and domain-specific information used to enrich and personalize responses.

**Database Type:** Relational Databases / Data Lakehouse Ideal for handling structured or semi-structured data with defined schemas and ACID compliance.

**Reason: Structure & Consistency** Well-defined schemas ensure reliable storage, consistency, and robust querying of user and operational data.

**Transaction Pattern:**

**Frequent Reads:** Repeated lookups of user profiles, preferences, or operational data for personalization and decision-making.

**Occasional Writes/Updates:** Periodic modifications to user information or metrics (e.g., profile changes, updated analytics).

**Concurrent Access:** Various services and user sessions accessing and updating structured data concurrently.

**Open-Source Examples:** MySQL or PostgreSQL (relational)**,** Apache Hudi or Delta Lake (data Lakehouse) [24]

**Semantic Context:**
Semantic context manages high-dimensional embeddings and vectorized data for similarity searches, enabling semantic understanding beyond keyword matching.
**Database Type:** Vector Databases Specifically designed for managing and querying high-dimensional vectors.
**Reason: Semantic Retrieval** Optimized for storing and retrieving vector embeddings, enabling efficient similarity searches and context-aware responses.
**Transaction Pattern:**
**Frequent Reads:** Frequent similarity searches on stored embeddings to retrieve contextually relevant information.
**Occasional Writes:** Inserting or updating embeddings when new data or content becomes available.
**Optimized for Vector Operations:** Specialized indexing (e.g., IVF, HNSW) to handle high-dimensional queries efficiently, with generally lower concurrency than chat-based contexts.
**Open-Source Examples:** Milvus [25]**,** FAISS (library, often embedded in custom solutions) [26]**,** Vespa [27]**,** pgvector extension [28]

## CONCLUSION:

In conclusion, by leveraging the distinct power of each context conversational, situational, and semantic GenAI applications can deliver responses that are both relevant and intuitive. Each context type requires specialized database solutions, ensuring not only the accurate storage and retrieval of information but also the ability to process data at scale. As we've seen, integrating the right combination of relational, document, and vector databases for each use case enables AI systems to offer more refined and contextually aware interactions. This strategic approach enhances the overall user experience, making AI-driven applications smarter, faster, and more reliable for end-users.

In conclusion, each type of context conversational, situational, and semantic plays a unique role in delivering comprehensive, accurate responses within GenAI applications. By leveraging the appropriate database solutions for each context, AI systems can combine real-time conversation logs, external domain data, and vectorized semantic knowledge into a cohesive and responsive experience. As we turn our attention to semantic context, we'll see how vector databases and similarity searches bring deeper meaning to user queries, further enhancing the quality and relevance of the AI's outputs.

## REFERENCES:


1. OpenAI, GPT-4 Technical Report, arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774.
2. Google DeepMind, Introducing Gemini: Multimodal AI by Google, 2023. [Online]. Available: https://blog.google/technology/ai/google-gemini-ai/.
3. Ouyang, L., et al., "Training language models to follow instructions with human feedback," Advances in Neural Information Processing Systems (NeurIPS), 2022. [Online]. Available: https://arxiv.org/abs/2203.02155.
4. P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," arXiv, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401.
5. E. Topol, "High-Performance Medicine: The Convergence of Human and Artificial Intelligence," Nature Medicine, vol. 25, no. 1, pp. 44–56, 2019. [Online]. Available: https://www.nature.com/articles/s41591-018-0300-7.
6. L. Chen et al., "AI for Financial Risk Management: Applications and Trends," Journal of Finance and Data Science, vol. 6, no. 1, pp. 1-16, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405918821000170.
7. A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," ICML, 2021. [Online]. Available: https://arxiv.org/abs/2103.00020.
8. M. Chen et al., "Evaluating Large Language Models Trained on Code," NeurIPS, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374.
9. J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, vol. 5, no. 2, pp. 265–277, 2019. [Online]. Available: https://arxiv.org/abs/1702.08734.

10. H. Jegou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," IEEE TPAMI, vol. 33, no. 1, pp. 117–128, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5432203.
11. J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74-80, 2013. [Online]. Available: https://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/.
12. R. Xin et al., "Deep Learning Model Versioning in Large-Scale AI Workflows," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2306.01588.
13. D. Crankshaw et al., "Clipper: A Low-Latency Online Prediction Serving System," NSDI, 2017. [Online]. Available: https://arxiv.org/abs/1703.06902.
14. M. Stonebraker and U. Çetintemel, "One Size Fits All": An Idea Whose Time Has Come and Gone, ICDE, 2005. [Online]. Available: https://ieeexplore.ieee.org/document/1427791.
15. R. Xin et al., "Deep Learning Model Versioning in Large-Scale AI Workflows," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2306.01588.
16. J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, vol. 5, no. 2, pp. 265–277, 2019. [Online]. Available: https://arxiv.org/abs/1702.08734.
17. P. Li, A. Ahmad, H. He, and X. Zhang, "Scaling AI Vector Search in Large-Scale Databases," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2305.12584.
18. H. Jegou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," IEEE TPAMI, vol. 33, no. 1, pp. 117–128, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5432203.
19. Redis, "Redis: In-Memory Data Structure Store," Redis Labs, 2024. [Online]. Available: https://redis.io/docs.
20. MongoDB, Inc., "MongoDB: NoSQL Database for Modern Applications," 2024. [Online]. Available: https://www.mongodb.com/docs/.
21. The PostgreSQL Global Development Group, "PostgreSQL Documentation," 2024. [Online]. Available: https://www.postgresql.org/docs/.
22. Oracle Corporation, "MySQL: Open-Source Relational Database System," 2024. [Online]. Available: https://dev.mysql.com/doc/.
23. The Apache Software Foundation, "Apache Hudi: Streaming Data Lake Platform," 2024. [Online]. Available: https://hudi.apache.org/docs/.
24. Milvus, "Milvus: Open-Source Vector Database for Scalable AI," 2023. [Online]. Available: https://milvus.io/docs.
25. Facebook AI Research, "FAISS: A Library for Efficient Similarity Search," 2023. [Online]. Available: https://faiss.ai/.
26. Vespa Team, "Vespa: A Scalable Open Source AI Search Engine," 2024. [Online]. Available: https://vespa.ai/.
27. pgvector, "pgvector: Open-Source Vector Similarity Search for Postgres," 2024. [Online]. Available: https://github.com/pgvector/pgvector.