ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

Bitezzy: A Recipe Recommendation Platform Powered by Vector Database

[1] Subhranil Chakraborty, [2] Wasiq Afnan Ansari, [3] Trisit Chanda, [4] Soudipta Sarkar, [5] Nabanita Nath, [6] Debasmita Mukherjee

[1] [2] [3] [4] [5] Master of Computer Applications, Techno Main Salt Lake, Kolkata, India
[6] Professor, Master of Computer Applications, Techno Main Salt Lake, Kolkata, India
Email: [1] subhranil.chak.sc@gmail.com, [2] wasiqafnan@gmail.com, [3] trisitchanda@gmail.com, [4] soudiptass@gmail.com, [5] nabanita315@gmail.com, [6] debasmita.wbut@gmail.com

***Abstract*— Bitezzy is an AI-driven culinary platform designed for intelligent recipe discovery, personalized recommendations, and subscription-based content access. The system combines a ReactJS front-end, a NodeJS backend, and MongoDB for persistent storage with an asynchronous processing pipeline and a vector database Qdrant for semantic retrieval. Recipe data and user interaction signals are converted into embeddings and indexed for similarity search, enabling recommendation beyond keyword matching. The platform supports recipe ingestion, update and delete synchronization, real-time query routing, and recipe-based as well as interaction-based recommendations. By combining semantic search with modular backend services, Bitezzy provides a scalable and responsive architecture for modern recipe recommendation systems.**

***Index Terms*— Recipe recommendation, vector database, semantic search, embeddings, recommendation system, web application.**

I. INTRODUCTION

Modern culinary applications must support more than simple recipe listing and keyword search. Users expect personalized suggestions, context-aware search, and fast responses even when the recipe catalog grows large. To address these requirements, Bitezzy adopts a modular architecture that com bines a web-based application stack with semantic retrieval and asynchronous background processing.

The system is built on a ReactJS frontend, a NodeJS backend, and MongoDB for persistent storage. To improve recommendation quality, the platform extends the core application with a worker-based pipeline and a vector database Qdrant. Recipe content, such as ingredients, cuisine type, preparation steps, and dietary labels, is transformed into embeddings so that similar recipes can be retrieved through semantic similarity rather than only through exact text matching.

The architecture of Bitezzy is organized around three main layers: user interaction, backend orchestration, and semantic retrieval. The frontend handles recipe submission, search, and browsing. The backend validates requests, routes them through the appropriate modules, and coordinates database operations. The recommendation layer stores embeddings in a vector database, which allows the system to retrieve recipes that are semantically close to a query, a user preference pattern, or a currently viewed recipe.

The system is extended with asynchronous processing so that heavy tasks do not block the main request–response cycle. When a recipe is added, updated, or deleted, the corresponding job is pushed to a background queue. A worker service processes the job, updates the primary database, generates or refreshes embeddings, and synchronizes the vector index. This design improves scalability, keeps the application responsive, and ensures that the recommendation engine remains consistent with the latest recipe data.

II. SYSTEM ARCHITECTURE AND RECIPE PROCESSING WORKFLOW

A. Recipe Ingestion and Indexing Flow

When a chef adds a new recipe, the recipe is first submitted through the Bitezzy application interface. The submitted data may include structured and unstructured fields such as the recipe title, ingredients, preparation steps, cuisine type, dietary labels, and associated media. Instead of sending this information directly into the recommendation layer, the system routes it through an asynchronous processing pipeline to keep the main request–response cycle lightweight and responsive.

After validation, the API pushes the recipe payload into a background queue managed by a job processor. A worker service continuously listens to the queue and consumes new jobs as they arrive. During processing, the worker may upload recipe images to cloud storage, extract image URLs, and prepare the dataset for downstream use. Once preprocessing is complete, the cleaned and structured recipe data is stored in MongoDB as the primary database.

Following storage, the system generates a semantic embedding for the recipe using an embedding model (openai/text embedding-3-large). The embedding captures contextual relationships among recipe attributes such as ingredients, cuisine style, and preparation techniques. This numerical representation is then indexed in the vector database along with metadata such as the recipe identifier,

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

cuisine type, dietary labels, and media references. Once indexing is complete, the recipe becomes available for semantic similarity search, personalized recommendations, and AI-driven discovery.

B. Update Recipe Flow

When an existing recipe is updated, the request is first sent to the API, which modifies the corresponding document in the primary database. To preserve consistency between the database and the semantic index, the update event is also added to the processing queue. The worker service then reprocesses the updated recipe, regenerates its embedding, and replaces the old vector in the vector database with the new one. This ensures that the recommendation system always reflects the most recent version of the recipe.

C. Delete Recipe Flow

When a recipe is deleted, the system performs a soft delete in the main database to maintain referential integrity. A delete job is then added to the queue for asynchronous cleanup. The worker processes this job and removes the corresponding vector from the vector database, ensuring that deleted recipes no longer appear in search results or recommendation lists.

D. Design Benefits

This architecture clearly separates ingestion, processing, persistence, and retrieval. By using asynchronous queues and worker-based processing, Bitezzy improves scalability and keeps the user experience responsive. The vector database layer enables semantic retrieval, allowing the platform to support recommendation tasks beyond simple keyword matching. The workflow also ensures that recipe additions, updates, and deletions remain synchronized across the application stack.

III. INTELLIGENT QUERY PROCESSING AND RECOMMENDATION RETRIEVAL

A. Query Submission and Controller-Level Processing

Bitezzy incorporates an intelligent query processing pipeline to handle diverse user requests efficiently. The process begins when a user submits a query through the client interface in the form of a chat message. This query may relate to recipe discovery, cooking guidance, or general conversational input. The backend controller first validates and formats the request before forwarding it to the orchestration layer for further processing.

This separation between request handling and decision making keeps the application modular and maintainable. It also allows the system to route user inputs to the most appropriate subsystem based on query intent and complexity.

B. Intent Determination and Route Classification

The orchestration layer analyzes the user query to determine intent. The system identifies whether the query is related to recipe search, cooking tips, or an unsupported category. After intent detection, a route classifier assigns the query to one of the predefined paths. This classification step enables the platform to dynamically route user requests to the correct module without manual intervention.

The routing mechanism also supports a cost-aware multi model strategy. High-complexity queries, such as semantic recipe search, are handled by stronger language models. Moderate queries, such as cooking tips, may be handled by mid-tier models. Off-topic or unsupported queries are processed using lightweight models that generate short, polite responses or redirections. This approach reduces operational cost while preserving response quality where it matters most.

C. Recipe Search Flow

When a query is classified as a recipe search request, it is forwarded to the recipe search module. The query is first normalized to improve compatibility with the embedding model. Normalization may include synonym expansion or language standardization. The processed query is then converted into a vector representation that captures semantic meaning rather than relying on exact keyword overlap.

The generated query embedding is passed to the vector database, where a similarity search is performed against stored recipe embeddings. The system identifies the most relevant recipes based on semantic closeness. After ranking the results, the corresponding recipe documents are fetched from the primary database, including metadata and media references. A language model then generates a structured natural language response and sends it back to the client.

D. Cooking Tips Flow

Queries classified as cooking tips are routed to a cooking assistance module. In this path, a moderately capable language model generates contextual advice tailored to the user's re quest. The generated guidance is returned directly to the user in a concise and helpful format.

E. Other Queries Flow

If the user query does not match any supported category, it is sent to the other queries module. A lightweight language model generates a polite decline or a redirection message that informs the user about the supported functionality of the platform. The response is then returned through the controller to the client.

F. Single Recipe Detail-Based Recommendation Flow

In addition to interaction-based recommendations, Bitezzy also supports a single recipe detail-based recommendation system. This feature helps users discover recipes similar to the one they are currently viewing. When a user opens a specific recipe, the system retrieves the full recipe information from the primary database, including the title, ingredients, preparation steps, cuisine type, dietary labels,

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

and related metadata.

The user can then view the complete recipe details in a structured format. From this information, the system extracts the most relevant attributes for similarity search, especially ingredients, dietary labels, and cuisine type. These attributes are combined into a search representation and converted into an embedding vector.

The resulting vector is used to perform a similarity search in the vector database. Since all recipes are already indexed as embeddings, the system can retrieve recipes that are semantically related in terms of ingredient composition, cuisine style, and dietary compatibility. The retrieved recipes are displayed as similar or recommended recipes on the recipe detail page, improving discovery and encouraging exploration of related dishes.

G. Summary of Retrieval Benefits

The combination of intent-aware routing, semantic recipe search, and recipe-to-recipe similarity search makes the recommendation system more flexible and user centered. By using embeddings and vector search rather than simple tag matching, Bitezzy can produce more meaningful recommendations and respond to a wider range of user needs in a scalable way.

IV. METHODOLOGY

A. System Design Overview

Bitezzy is designed as a modular recipe recommendation platform that combines a web application stack with semantic retrieval and asynchronous processing. The architecture separates user interaction, backend orchestration, and recommendation retrieval into distinct components so that each part of the system can scale independently. The frontend handles user requests such as recipe submission, recipe browsing, and chat-based queries. The backend validates requests and forwards them to the appropriate processing pipeline. The recommendation engine uses vector representations of recipes and user preferences to support semantic search and personalized suggestions.

B. Query Processing and Routing

The system incorporates an intelligent query processing pipeline to handle user requests efficiently. When a user submits a chat message, the backend controller first validates the request and then forwards it to the orchestration layer. The orchestration module identifies the intent of the query and routes it to one of three supported paths: recipe search, cooking assistance, or general response handling.

A route classifier is used to categorize the query according to its semantic purpose. Recipe-related queries are processed through a similarity search pipeline. Cooking-tip queries are handled by a lightweight generation path, while unsupported queries are redirected through a polite fallback response. This routing strategy improves both response quality and computational efficiency by avoiding unnecessary use of high cost models for simple tasks.

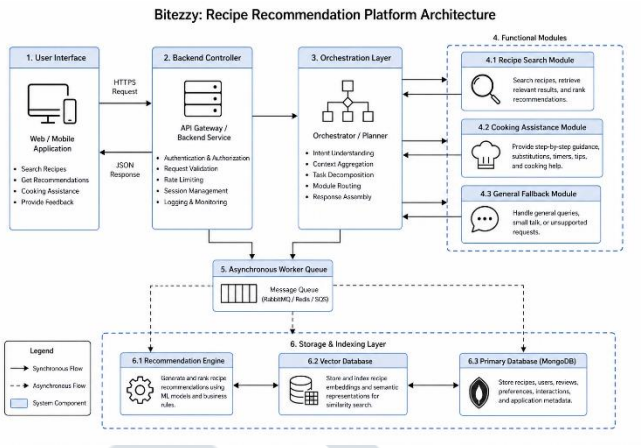

Figure 1. Overall Bitezzy chatbot and orchestration flow.

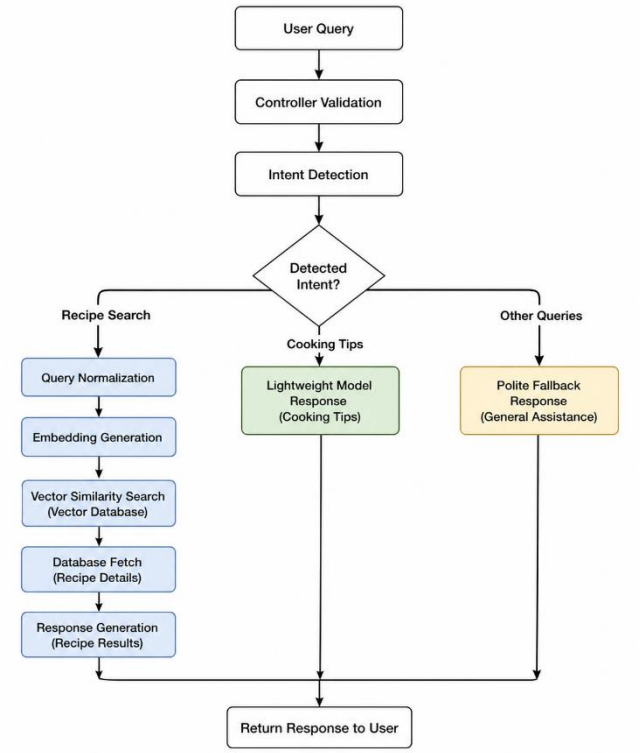

Figure 2. Query processing and routing workflow for recipe search, cooking tips, and fallback responses.

C. Recipe Ingestion and Vector Indexing

Recipe ingestion begins when a chef submits a new recipe through the application interface. The submitted content may include a title, ingredients, preparation steps, cuisine type, dietary labels, and associated images. Instead of processing the recipe synchronously, the request is added to a background queue so that the main request–response cycle remains lightweight.

A worker service consumes jobs from the queue and performs preprocessing tasks such as image upload, URL extraction, and data normalization. After preprocessing, the

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

structured recipe is stored in the primary database. The system then generates an embedding from the recipe description and metadata. This embedding encodes semantic relationships among recipe attributes, enabling similarity-based retrieval beyond keyword matching. The generated vector is indexed in the vector database together with metadata such as recipe title, cuisine type, dietary labels, and media references.

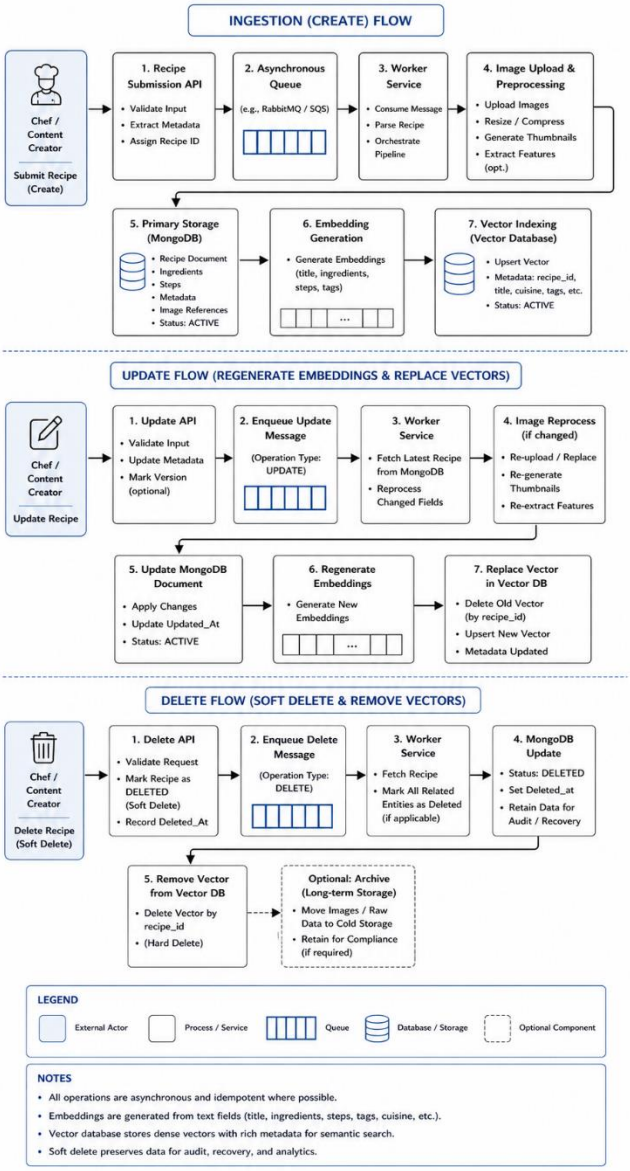

Figure 3. Recipe ingestion, update, delete, and semantic retrieval pipeline.

D. Update and Delete Synchronization

When a recipe is updated, the corresponding document is modified in the primary database and an update job is added to the queue. The worker reprocesses the updated recipe, regenerates its embedding, and replaces the old vector in the vector database. This ensures that semantic search always uses the latest recipe representation.

For deletion, the system performs a soft delete in the main database and adds a delete job to the queue. The worker then removes the associated vector from the vector database so that deleted recipes no longer appear in search or recommendation results. This synchronization preserves consistency across storage layers and keeps the recommendation engine aligned with the current catalog.

E. Recommendation Generation

The recommendation flow uses both user interaction signals and recipe similarity. Recent user interactions, such as viewed recipes, are summarized using lightweight metadata that includes cuisine type and dietary labels. This compact preference representation is converted into an embedding and used as the query vector for semantic retrieval. The vector database returns the nearest recipe embeddings, and the matched recipe documents are fetched from the primary database. The response is then presented to the user as recommended recipes. A similar process is also used when a user opens a single recipe detail page, where the current recipe attributes are transformed into a search vector to retrieve closely related recipes.

F. Design Rationale

The proposed architecture is built around three goals: scalability, responsiveness, and semantic relevance. Asynchronous queues prevent heavy processing from blocking the user inter face. Worker-based execution makes ingestion and indexing reliable. Vector search improves recommendation quality by capturing semantic similarity among recipes rather than re lying only on exact tag overlap. These design choices make Bitezzy suitable for real-time, AI-powered culinary discovery.

V. RESULT AND DISCUSSION

A. System Implementation Results

The proposed Bitezzy platform was implemented as a modular recipe recommendation system with three main operational capabilities: recipe ingestion and indexing, intelligent query routing, and semantic recommendation retrieval. The architecture successfully separates user-facing requests from heavy background processing, allowing recipe creation, update, and delete operations to be handled asynchronously through a worker-based queue. This design keeps the main application responsive while ensuring that the vector database remains synchronized with the primary database.

B. Recipe Ingestion and Indexing Behavior

When a new recipe is submitted, the system first stores the structured recipe data in the primary database and then generates an embedding for semantic indexing. This approach enables the platform to represent each recipe not only by explicit metadata such as cuisine type and dietary labels, but also by semantic content derived from the

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

ingredients and preparation details. As a result, recipes can be retrieved later through similarity-based search even when the query does not contain the exact same words used in the source recipe.

The use of asynchronous job processing improves system responsiveness by preventing embedding generation and vector indexing from blocking the user request. This is especially useful when recipe content includes images, which may require additional preprocessing before storage and indexing.

C. Query Routing and Response Generation

The intelligent query processing pipeline improves the flexibility of the system by directing each user query to the appropriate module. Recipe-related queries are handled through semantic retrieval, cooking-tip queries are routed to a lightweight assistance path, and unsupported queries are redirected through a fallback response. This classification mechanism reduces unnecessary processing and allows the system to balance response quality with computational efficiency.

A notable advantage of this design is that the system does not treat all user requests in the same way. Instead, it applies a cost-aware routing strategy that reserves stronger language model-based generation for tasks that require richer contextual responses, while simple or off-topic queries are handled with smaller and faster response paths.

D. Recommendation Quality and Semantic Retrieval

The main recommendation advantage of Bitezzy comes from its use of vector embeddings and semantic similarity search. Traditional recipe recommenders often rely on tag overlap or keyword matching, which can miss relevant recipes when the vocabulary differs even though the culinary meaning is similar. In contrast, the vector database representation allows the system to retrieve recipes that are related in terms of ingredients, cuisine style, and dietary compatibility.

This approach is also effective for single recipe detail-based recommendations. When a user opens one recipe, the system extracts the most relevant recipe attributes and uses them as a similarity query. The retrieved results are therefore not limited to recipes with identical titles or tags, but instead include dishes that are semantically close to the current recipe. This improves content discovery and creates a more context-aware browsing experience.

E. Update and Delete Consistency

The update and delete flows ensure that recommendation results remain accurate over time. When a recipe is updated, its embedding is regenerated and the previous vector is replaced. When a recipe is deleted, the corresponding vector is removed from the semantic index after the database performs a soft delete. These synchronization steps prevent stale records from appearing in search or recommendation results.

This consistency layer is important in production systems because recommendation quality depends on the alignment between the stored recipe content and the indexed semantic representation. Without this synchronization, users could receive outdated or irrelevant recommendations.

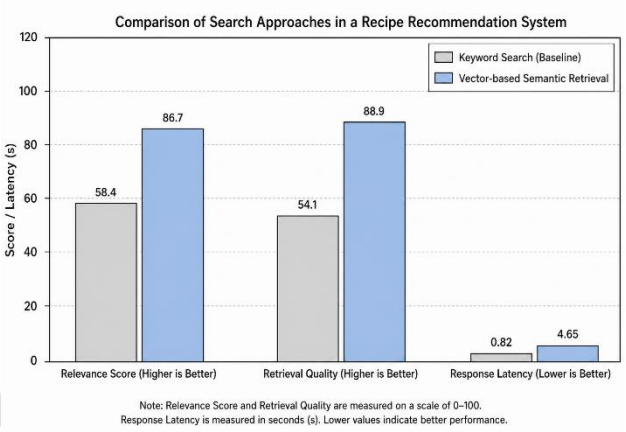

Figure 4. Comparison of keyword search and vector-based semantic retrieval.

F. Discussion

The proposed system demonstrates that a vector database can serve as an effective semantic retrieval layer for recipe recommendation. Compared with a purely keyword-driven design, the embedding-based approach provides better representation of culinary meaning, making it easier to recommend recipes that match user intent even when surface terms differ. The architecture also supports scalability through background workers and queue-driven processing, which makes it suitable for a growing recipe catalog.

From a system design perspective, the combination of asynchronous processing, semantic retrieval, and modular query routing provides a practical balance between performance and user personalization. The platform is therefore not limited to recipe search alone; it also supports chat-based assistance and recipe similarity discovery within the same architecture.

G. Suggested Evaluation Figure

A useful figure for this section would be a comparison chart showing keyword-based search versus vector-based search in terms of relevance, response time, and retrieval quality.

H. Limitations

The current implementation focuses on system design and semantic retrieval behavior. A fuller empirical evaluation would benefit from quantitative metrics such as retrieval ac curacy, latency, and user satisfaction. Future experiments may compare the proposed vector-based approach against baseline keyword matching and rule-based recommendation methods using a controlled test dataset.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 07, July 2026

VI. CONCLUSION

Bitezzy presents a modular recipe recommendation platform that combines asynchronous processing, semantic embeddings, and vector database retrieval to improve recipe discovery and personalization. By separating recipe ingestion, update, and delete operations from the main request–response cycle, the system remains responsive while maintaining consistency between the primary database and the semantic index.

The proposed architecture also supports intelligent query routing, allowing recipe search, cooking assistance, and fall back handling to be processed through different paths based on user intent. In addition, the single recipe detail-based recommendation flow enables the system to surface semantically related recipes from the currently viewed item, improving the browsing experience and increasing the relevance of suggestions.

Overall, the design demonstrates that vector-based retrieval is a practical and scalable approach for modern culinary applications. Future work may include richer user profiling, improved ranking strategies, and evaluation using larger datasets and quantitative recommendation metrics.

VII. ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to all those who directly or indirectly contributed to the successful completion of this project, *Bitezzy: Recipe Finder & Recommendation System.*

We are especially grateful to our project guide, Prof. Debasmita Mukherjee, MCA Department, Techno Main Salt Lake, for her invaluable guidance, encouragement, and continuous support throughout this work. Her insightful feedback and technical suggestions helped us refine our ideas and complete the project in a structured manner.

We also thank the faculty members of the MCA Department, Techno Main Salt Lake, for providing the academic foundation, resources, and supportive environment needed to carry out this work. Our sincere thanks are also due to the administration and management of Techno Main Salt Lake for the facilities and infrastructure that made this project possible.

We acknowledge the effort and dedication of our team members**, Subhranil Chakraborty, Wasiq Afnan Ansari, Trisit Chanda, Soudipta Sarkar,** and **Nabanita Nath**, whose contributions across frontend, backend, and GenAI development were essential to the completion of this project.

Finally, we are deeply grateful to our families and friends for their patience, encouragement, and constant moral support throughout the duration of this work

REFERENCES

[1] Qdrant, “Qdrant vector database,” 2025. [Online]. Available: https://qdrant.tech/

[2] MongoDB, “MongoDB documentation,” 2025. [Online]. Available: https://www.mongodb.com/docs/

[3] BullMQ, “BullMQ documentation,” 2025. [Online]. Available: https:// docs.bullmq.io/

[4] OpenAI, “Embeddings guide,” 2025. [Online]. Available: https:// platform.openai.com/docs/guides/embeddings

[5] LangChain, “LangGraph documentation,” 2025. [Online]. Available: https://langchain-ai.github.io/langgraph/