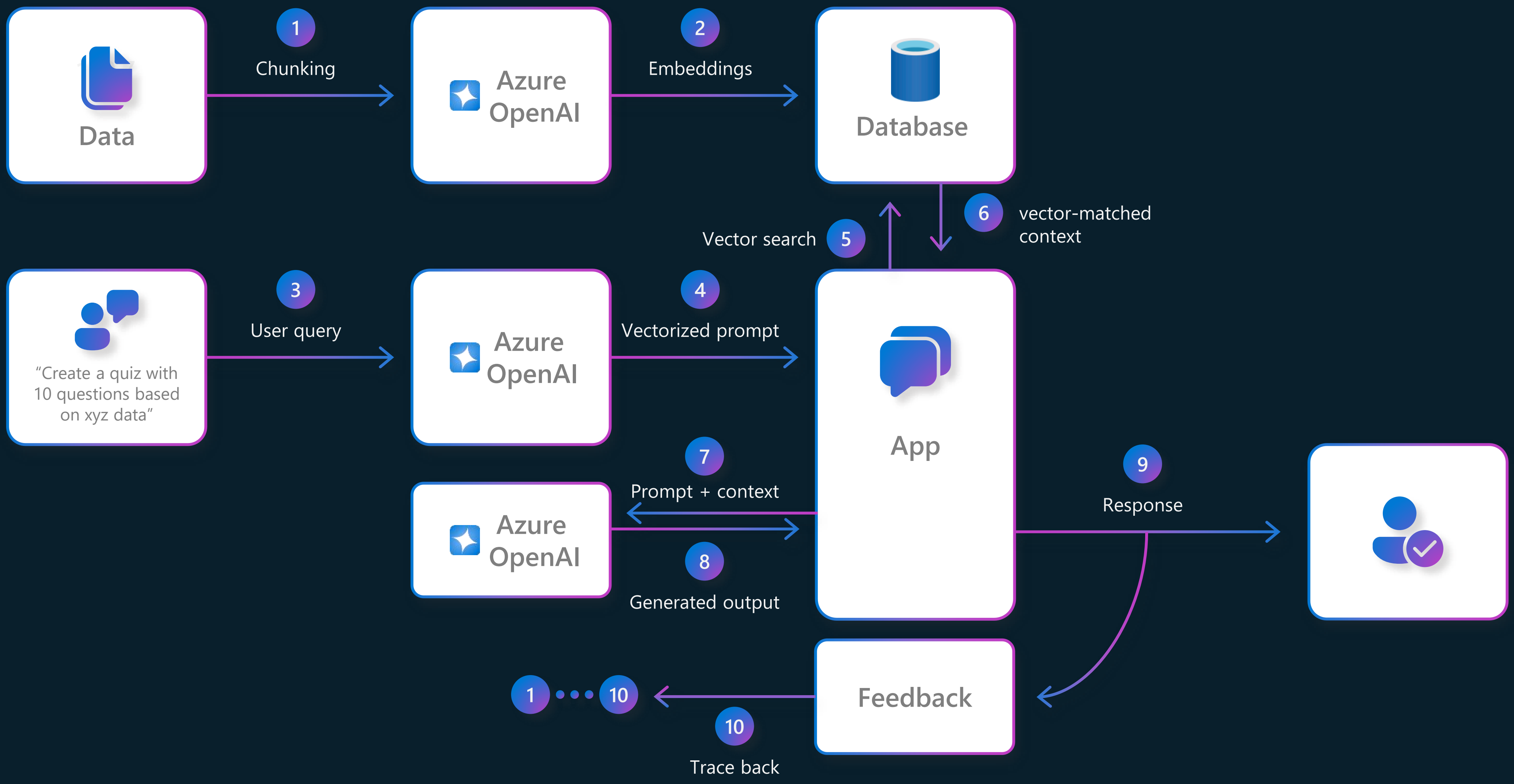
Better accuracy and scale with GraphRAG and OmniRAG

Basic RAG (Retrieval Augmented Generation)



Examples of problems in AI responses*

You: how many bikes of each type do you have in your shop?

Bot: there are two bikes – one mountain bike and one road bike

Too few records and no aggregates fetched from the vector store

You: what is total order amount for the road bikes with white frame?

Bot: The records provided don't contain this information

No aggregates fetched from vector store

You: how much was my last transaction?

Bot: Looks like your last purchase order of $100 was made a year ago

Stale and unsorted data fetched from vector store

* - Using standard AdventureWorks/CosmicWorks dataset

Basic RAG is NEVER enough!

Only yields portion of the source data without aggregates/projections -> lower accuracy of AI solution

Motivates to use all available model context window for (possibly irrelevant) custom data -> higher cost for inferencing

Requires generating embeddings on the entire dataset -> higher cost for compute/storage

Requires pre-population of semantic index (facets/column-level stats) by guessing what the questions will be -> lower accuracy, higher costs for search engine

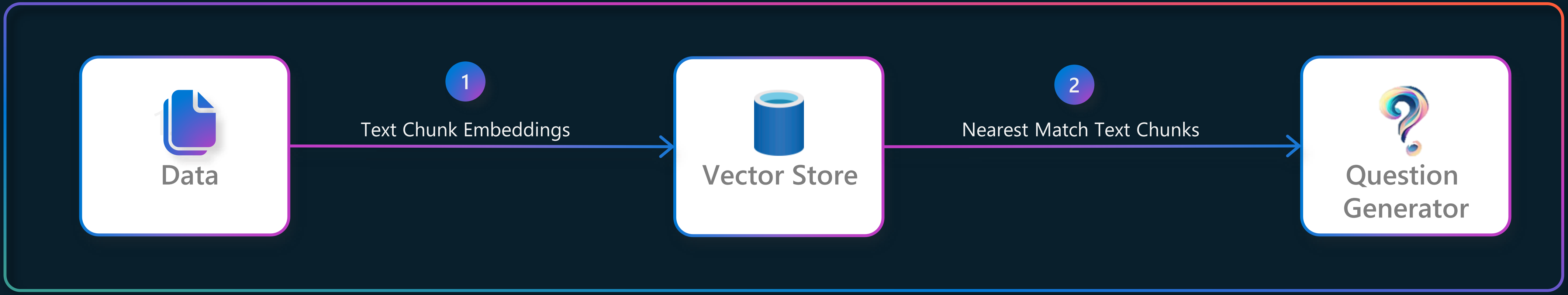
How to improve RAG

- Most popular improvement is semantic reranking on top of vector search results
  - Can only produce marginally better results
  - Most context is already ignored by the time reranking starts
- Less popular (but more impactful) improvement is hybrid search - combining semantic search with full-text search with subsequent re-ranking
  - Much better semantically relevant results
  - Still cannot solve nuanced/multi-hop queries
- The only way to solve the accuracy is to “understand” the full context
  - The first and mandatory step is to create a connected model of the entire corpus of data

Why Knowledge Graph?

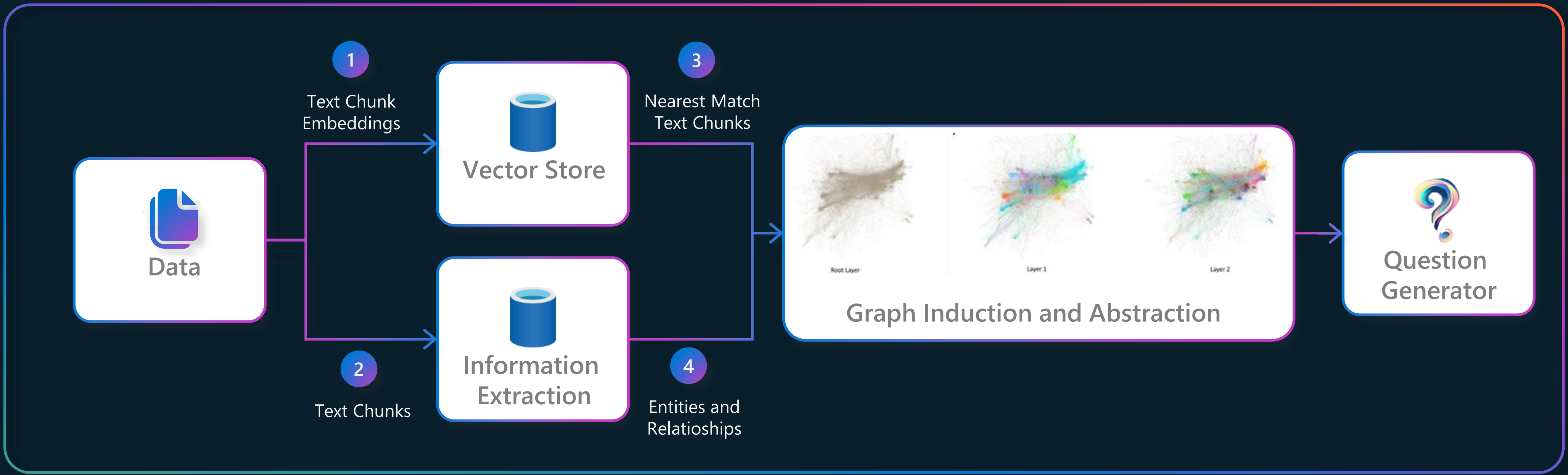
- Better at handling complex and multi-hop queries
    - Provides more complete and contextually rich answers
- Provides grounding source and chain of thoughts
    - Enhances the traceability/audit of the responses
    - Supports hallucination checking
- Particularly effective for tasks that require reasoning across many documents/stores
    - Explores adjacent topics
    - Answers nuanced questions
- Bonus: saves some AI costs as it contains curated/prepped data structure, immediately usable by LLM/SLM

Preprocessing Pipeline – Basic RAG



- ✅ Information lookup that is too big for context window
- ❌ Questions that require multi-hop reasoning
- ❌ Topics that have thin connections across many documents
- ❌ Abstractions and thematic retrievals

Preprocessing Pipeline – GraphRAG



- Information lookup that is too big for context window
- Questions that require multi-hop reasoning
- Topics that have thin connections across many documents
- Abstractions and thematic retrievals

Vector Search Results

Real-world example with lawsuits dataset

Recall = 10%

	Legal Case	Relevant
1	United Mutual Savings Bank v. Riebli (1960)	True
2	Wilkening v. State (1959)	False
3	Wilber Development Corp. v. Les Rowland Constr., Inc. (1974)	False
4	Washington Hydroculture, Inc. v. Payne (1981)	False
5	United Mutual Savings Bank v. Riebli (1960)	False
6	Strickland v. City of Seattle (1963)	False
7	King County v. Boeing Co. (1963)	False
8	DeHoney v. Gjarde (1925)	False
9	Evans v. Seattle (1935)	False
10	Arnold-Evans Co. v. Hardung (1925)	False

Semantic Ranker Results

Real-world example with lawsuits dataset

Recall = 20%

	Legal Case	Relevant
1	United Mutual Savings Bank v. Riebli (1960)	True
2	Javins v. First Nat'l Realty Corp (1970)	True
3	Geise v. Lee (1975)	False
4	McCutcheon v. United Homes Corp. (1971)	False
5	Reardon v. Shimelman (1925)	False
6	Woldson v. Woodhead (2005)	False
7	Donworth v. St. Paul Etc. R. Co. (1905)	False
8	Senske v. Washington Gas & Elec. Co. (1931)	False
9	Evans v. Seattle (1935)	False
10	Miller v. Vance Lumber Co.(1932)	False

GraphRAG Search Results

Real-world example with lawsuits dataset

Recall = 70%

Legal Case	Relevant
United Mutual Savings Bank v. Riebli (1960)	True
Javins v. First Nat'l Realty Corp (1970)	True
Foisy v. Wyman (1973)	True
Thomas v. Housing Authority (1967)	True
Jorgensen v. Massart (1963)	True
Martindale Clothing Co. v. Spokane & Eastern Trust Co. (1914)	True
Stuart v. Coldwell Banker (1987)	True
Papac v. City of Montesano (1956)	False
Finley v. City of Puyallup (1957)	False
Bach v. Sarich (1968)	False

Basic RAG vs GraphRAG

	No Augmentation	Basic RAG	GraphRAG
Context Awareness	None	Some	Deep
“Direct hit” search results	✘	✔	✔
Topically relevant connections	✘	✘	✔
Deep and sparse connections	✘	✘	✔
“Question behind the question”	✘	✘	partial
Hallucination Risk	High	Low	Very Low
Supports Hallucination Checking	✘	✔	✔
Provides source references	✘	✔	✔
Answers direct questions	✘	Usually	✔
Answers nuanced questions	✘	✘	✔
Explores adjacent topics and multiple perspectives	✘	✘	partial
Reasons across many documents	✘	~10	✔
Time to response	very fast (~5s)	fast (~5-10s)	medium (~10-30s)

Microsoft Vector Database Offerings

Offering	Capabilities	Maturity Level	Vector Data Type	Functions & Features	Indexing Algorithms	Integration Options
Azure SQL Database	Native vector data type (VECTOR), similarity search, hybrid search	**Preview**	VECTOR(n) (binary format)	VECTOR_DISTANCE VECTOR_SEARCH VECTOR_NORM VECTOR_NORMALIZE VECTORPROPERTY	Approximate vector index and vector search are in preview and currently only available in SQL Server 2025 (17.x) Preview.	Azure OpenAI, LangChain, Semantic Kernel, EF Core
SQL Server 2025	On-premises support for vector search and indexing	**Preview**	VECTOR(n)	Same as Azure SQL	DiskANN (ANN), k-NN *(Preview)*	Azure OpenAI, JSON casting
Azure Database for PostgreSQL (Flexible Server)	Integrated vector store using pgvector extension	**GA**	vector (via pgvector)	Similarity search using cosine, dot product, Euclidean	DiskANN, HNSW, IVFFlat	Azure OpenAI, Hugging Face
Azure Cosmos DB (NoSQL)	Integrated vector search with multimodal support	**GA**	Native vector fields	Cosine, dot product, Euclidean distance	DiskANN, HNSW	Azure OpenAI, RAG pattern, prompt engineering
Azure Cosmos DB for MongoDB (vCore)	Vector search via MongoDB vector capabilities	**GA**	Native vector support	Similarity search	DiskANN, IVFFlat	MongoDB tools, Azure OpenAI
Azure Cosmos DB for PostgreSQL	PostgreSQL-compatible vector search	**GA**	pgvector	Same as PostgreSQL	DiskANN, IVFFlat	Hugging Face, Azure OpenAI
Microsoft Fabric (SQL + Spark)	SQL engine supports vector search; Spark supports libraries like FAISS	**Preview / Emerging**	SQL: VECTOR(n); Spark: via Python libraries	SQL: same as Azure SQL; Spark: FAISS, Scikit-learn	DiskANN (SQL), FAISS (Spark)	Azure OpenAI, LangChain, RAG pipelines
Azure AI Search	Dedicated vector index with hybrid search and reranking	**GA**	Internal vector format	Semantic ranking, hybrid search	IVF, quantizedFlat, DiskANN	Azure OpenAI, Cognitive Search

Full Text Search & Ranking

- Search for text in documents
- Improved search relevancy and efficiency with a Full Text Index
- Rank text search results by relevancy (e.g. with BM25)

Full Text Search

Quickly and efficiently find keywords and terms in properties

Text Analyzers

Tokenization
Stopword removal
Stemming

Text Ranking

Find the documents most relevant to your search

Full Text Search

Performance optimizations

Reduced search latency and RU charge

Multi-language support

Efficient text search in any language

Phrase search

Find data items with entire phrases "Cosmos DB and Vector Search" rather than individual terms

Fuzzy search

Error correction: "Cossmos DB" vs "KosmoDB" vs "Cosmos DB"

Hybrid Search for improved search relevancy

Reciprocal Rank Fusion (RRF)

Vector Similarity Score + Full Text Scoring (BM25) = More Relevant Search Results

Example Query

```
SELECT TOP 10 *
FROM c
ORDER BY RANK RRF(FullTextScore(c.text,['keyword1','keyword2']), VectorDistance(c.vector, @vector))
```

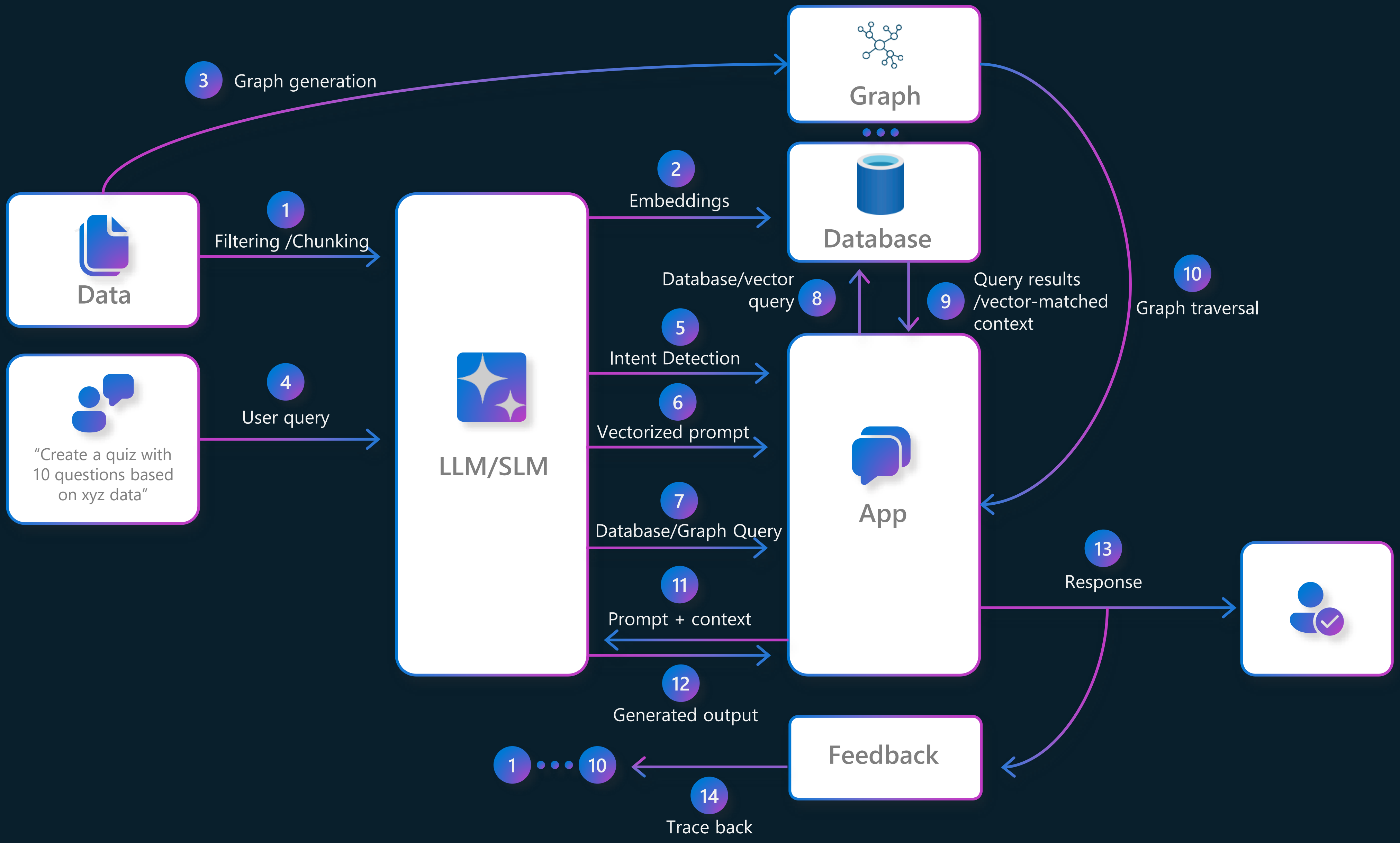
How to generate a graph

- From structured data - data wrangling
  - Create triples (subject/predicate/object) from well-known structured data
  - [Optional] Create structured data out of unstructured (e.g. with Azure AI Document Intelligence)
- From unstructured data - GraphRAG Indexing API
  - Use GraphRAG Query API for querying with NL
  - [Optional] Use graphml2ttl.py script to convert GraphRAG output to Turtle format and load Turtle-formatted graphs to an RDF Triple Store

Are we done yet?

- Now that we have a graph of our data, what else do we need for fast and accurate context retrieval?
  - Graph IS NOT an appropriate data structure for some types of data (e.g. images, time series, transactions, etc.), so it's necessary but not enough by itself
  - Generating the graph for some datasets/data sources is not feasible
  - There always will be sources of data available for querying but not full export and conversion for various reasons (CRM, ERP, HRM, cache, data lakes, data warehouses, etc.)
- MULTIPLE SOURCES OF DATA MUST BE USED IN CONTEXT RETRIEVAL (not just vector or graph or some other uber-store), so virtualization of context retrieval across multiple sources (including graph) is needed
  - Preferably seamless to the user (developer retrieving the context)
  - Preferably leveraging the results from one source to retrieve better context from others
  - Graph can still be (and better be) used to connect them all through hyperlinks.

OmniRAG Pattern



OmniRAG Core Tenets

- **Omni-source with data virtualization**
  - Not limited to vector store, utilizing ALL data sources that can bring value to the context for AI
  - Use data wherever it is, in the original format, minimizing data movement and transformation
- **Knowledge graph**
  - Contains entities/relationships from existing data to make it readily available for AI to reason over (along with original data)
- **User intent detection**
  - Allows automatic routing of user's query to the right source, leverages AI
- **Runtime NL2Query conversion**
  - Convert user query to the source's query language using simple utterance analysis and/or AI
- **Session analytics**
  - Required to fine-tune golden dataset of questions + intent so intent detection improves. Could be used for semantic cache generation/curation

Basic RAG vs OmniRAG

Rough subjective comparison

	Basic RAG	OmniRAG
Sources	Vector	Graph, Vector, Database, REST API, files, etc.
Intent detection	N/A	Yes
Semantic model generation	Generic (embeddings)	Custom (with code) or with LazyGraphRAG
NL2Query	N/A	Yes (built-in + code)
Global Search	No	Yes
Scalable	No	Yes (by leveraging native scaling of each source)

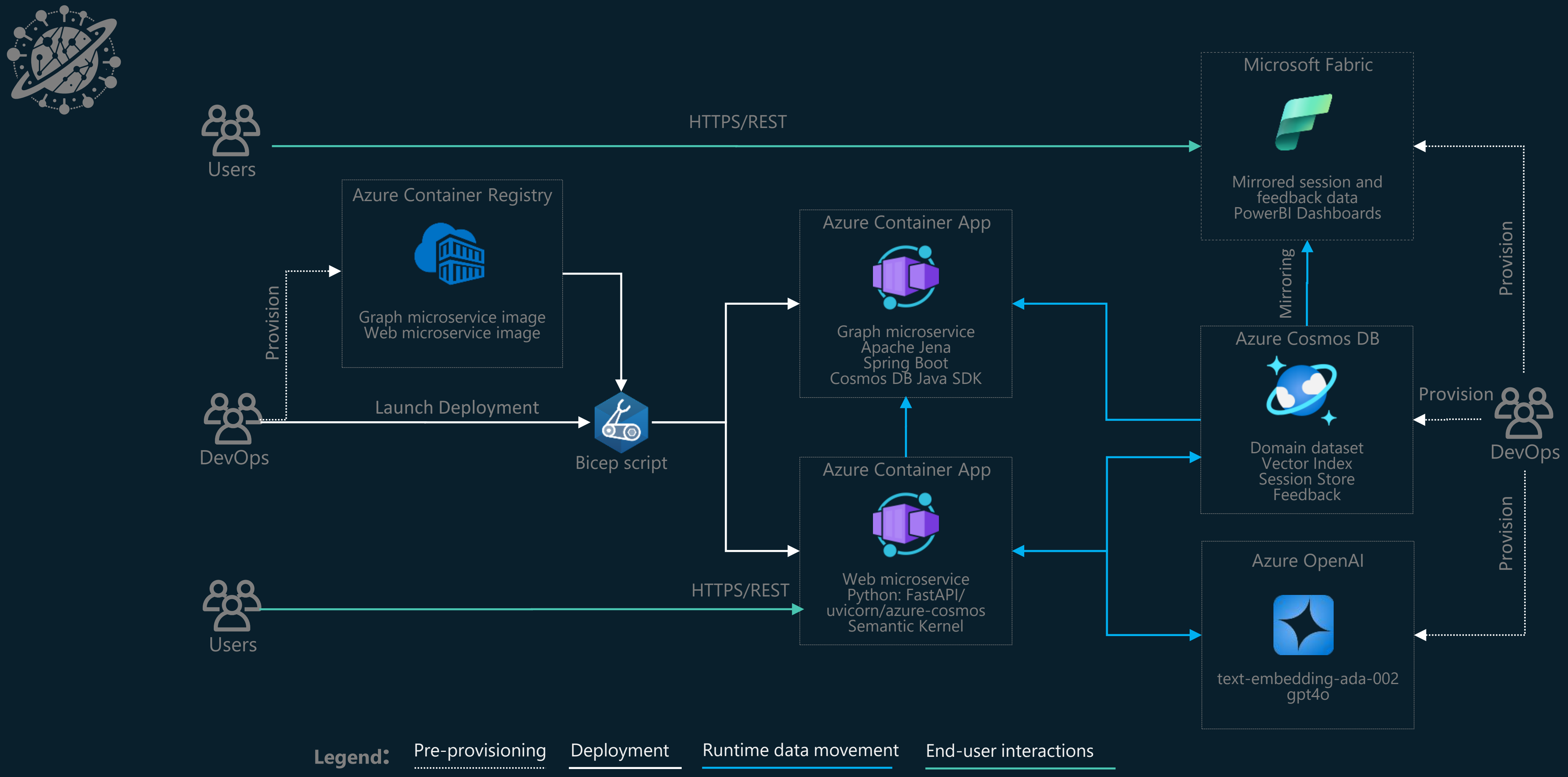
CosmosAIGraph as OmniRAG implementation

aka.ms/caig

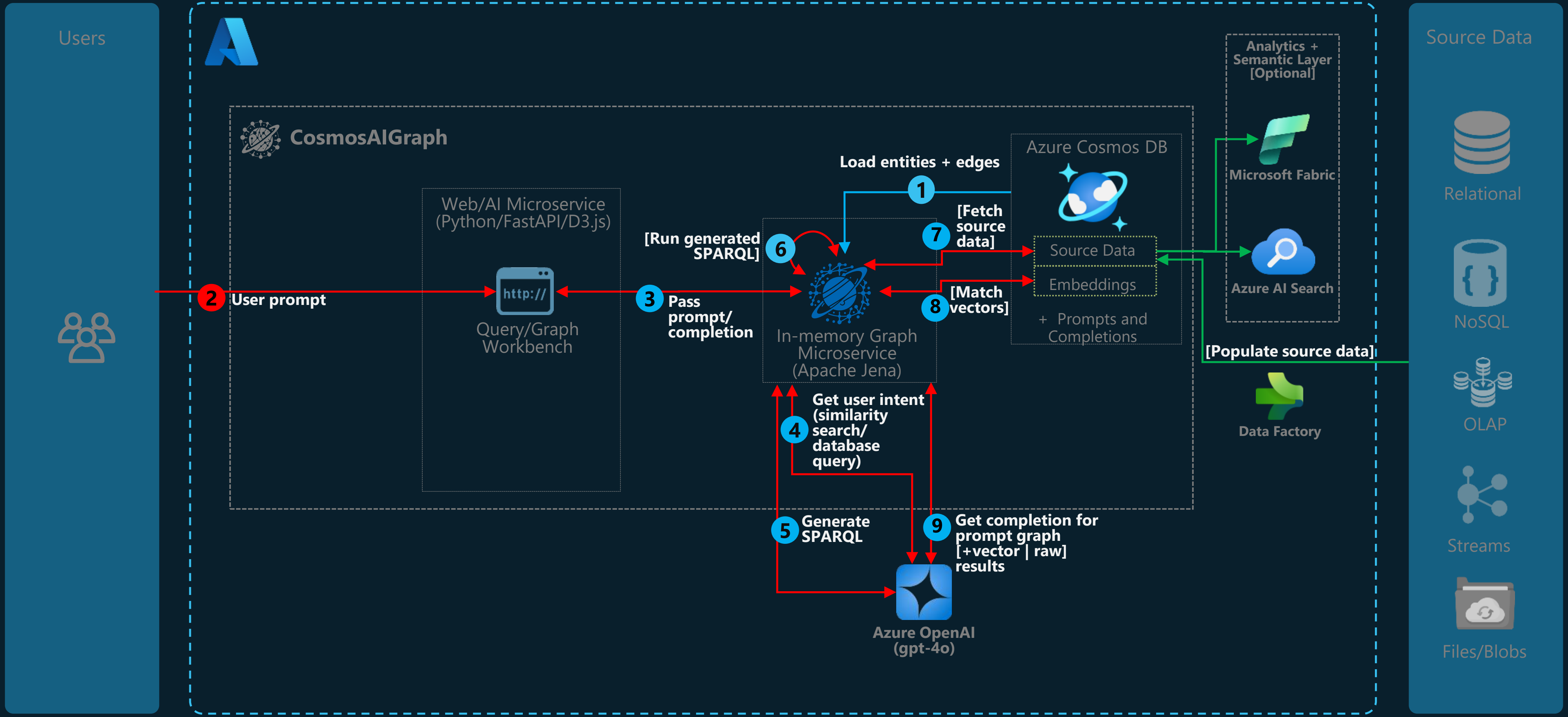
What is it providing?

- OmniRAG pattern reference implementation using knowledge graph/vector database/raw database
- Knowledge graph construction from structured data (Azure Cosmos DB, TTL file)
- User intent detection powered by built-in keyword scan and AI
- Scalable indexed RDF triple store database with in-memory model and persistent store for graph and domain data (Azure Cosmos DB)
- “Natural language to graph query” generator powered by AI with optional feedback loop

CosmosAIGraph Deployment Architecture



CosmosAIGraph Solution Architecture



Legend: Primary Flow (hot path) Background Flow (cold path) Data Ingestion/Mirroring [Optional actions] 1 User actions 2 Service actions

Public GitHub repo: aka.ms/caig

CosmosAIGraph Summary

- It's a reusable reference design and implementation. **It's not a product**
- It is built on the following:
  - ✓ **Cosmos DB for NoSQL** service, natively supporting vector search (DiskANN)
  - ✓ **RDF** technologies – triples, ontologies, SPARQL queries
  - ✓ **Apache Jena** – in-memory indexed RDF graph for fast performance and low costs
  - ✓ **Python 3** – UX, DAL and orchestration with fastapi, pydantic, azure-cosmos
  - ✓ **Azure OpenAI** service with industry-leading LLMs
  - ✓ **Semantic Kernel** – for pluggable and extensible orchestration
- Designed as set of microservices
- Deployed to Azure Container Apps with Bicep

Call to Action!

- ✓ Go to public repo aka.ms/caig and explore
- ✓ Deploy it to your Azure subscription
- ✓ Give us your feedback on OmniRAG and CosmosAIGraph!