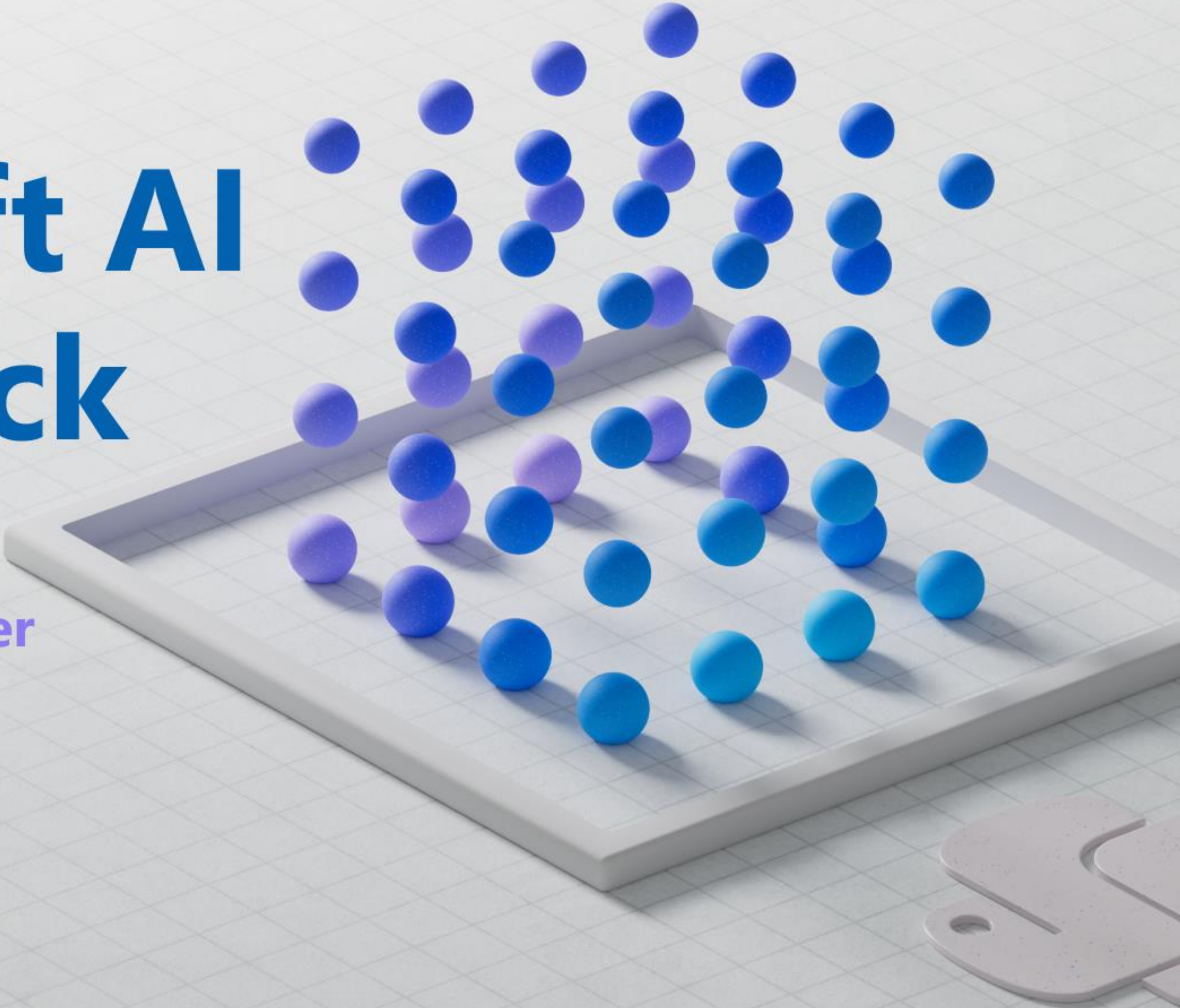
January 29th - February 12th

The Microsoft AI Chat App Hack

Build, innovate, and #HackTogether

aka.ms/hacktogether/chatapp



Microsoft

The AI Chat App Hack

January 29th - February 12th

29th ▶
Building a RAG Chat App in Python

30th ▶
Customizing your RAG Chat App

31st ▶
Azure AI Search Best Practices

1st ▶
GPT-4 with Vision

2nd
HACK
HACK
HACK

3rd
HACK
HACK

4th
HACK
HACK
HACK

5th ▶
AM: RAG Chat Web Components

PM: Access Control in RAG Chat Apps

6th ▶
Evaluating a RAG Chat App

7th ▶
RAG Chat Special Topic

8th ▶
Continuous Deployment of your Chat App

9th
HACK
HACK

10th
HACK

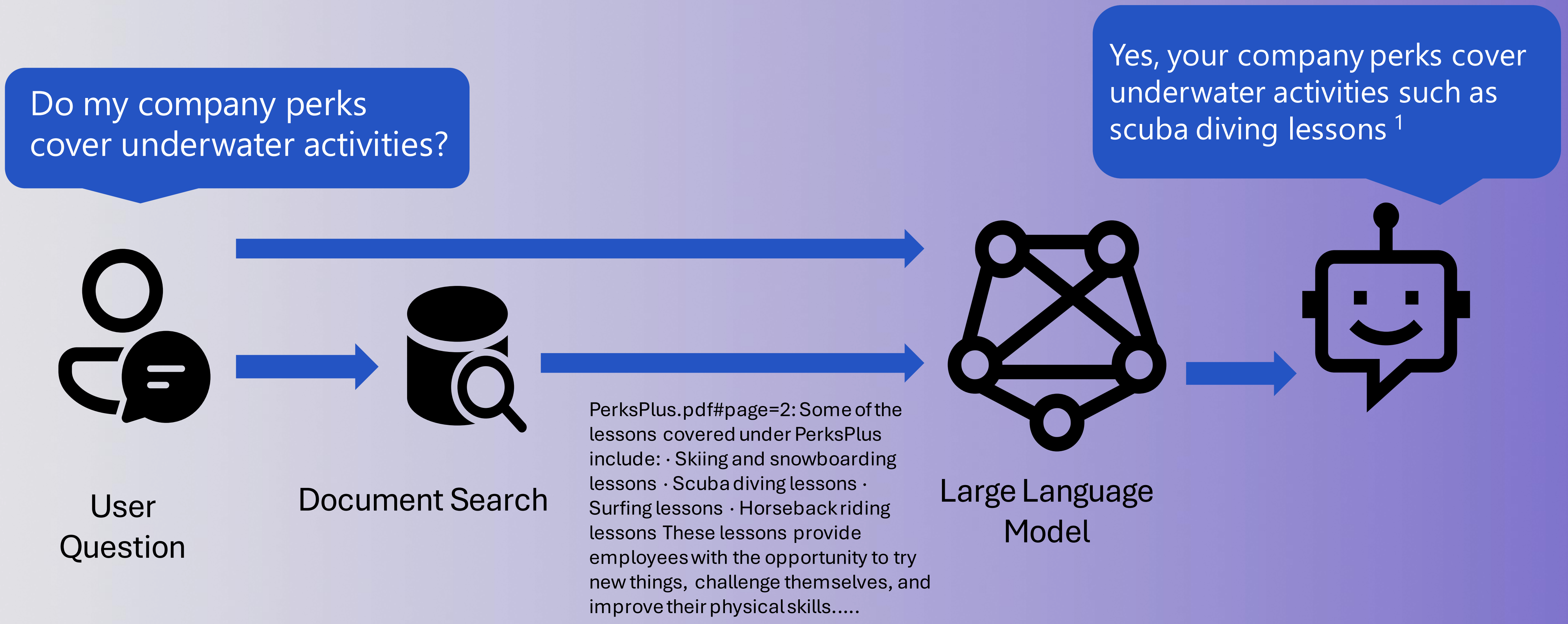
11th
HACK
HACK

12th
SUBMIT YOUR PROJECT

Build, innovate, and #HackTogether

Azure AI Search
Best Practices for RAG

RAG: Retrieval Augmented Generation



Do my company perks cover underwater activities?

User Question

Document Search

PerksPlus.pdf#page=2: Some of the lessons covered under PerksPlus include: · Skiing and snowboarding lessons · Scuba diving lessons · Surfing lessons · Horseback riding lessons These lessons provide employees with the opportunity to try new things, challenge themselves, and improve their physical skills.....

Large Language Model

Yes, your company perks cover underwater activities such as scuba diving lessons [1]

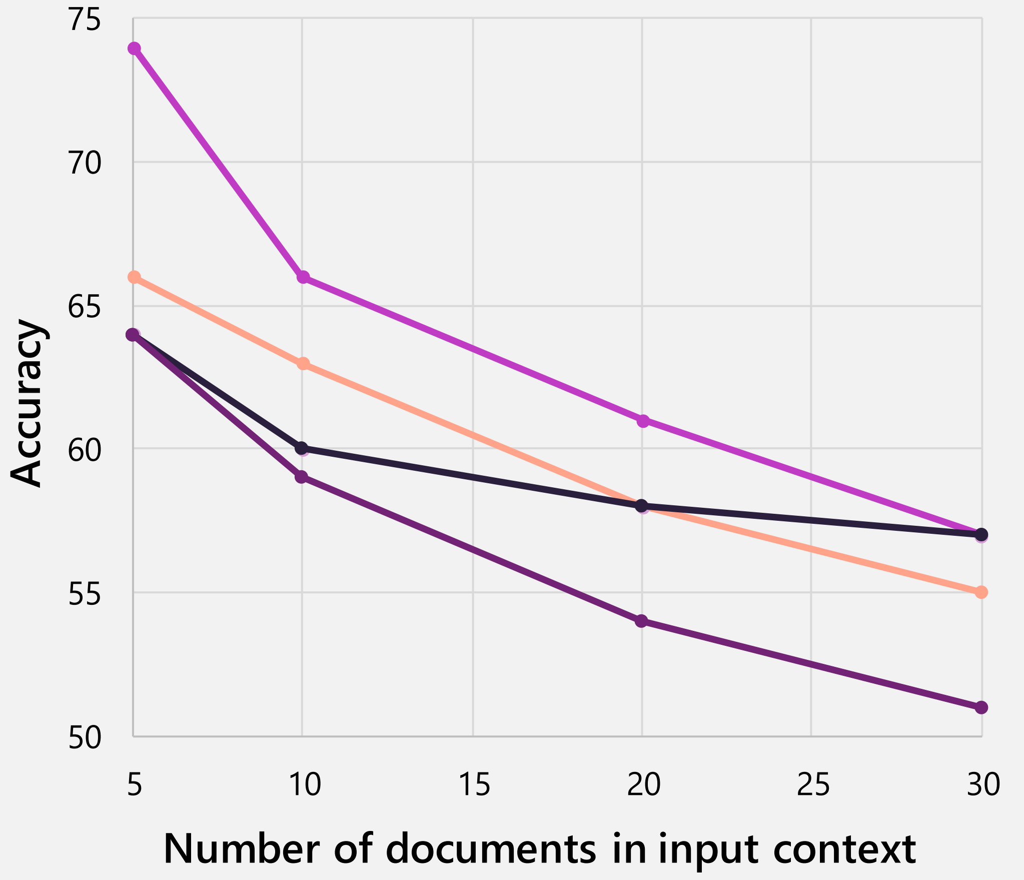
Microsoft

Robust retrieval for RAG chat apps

- Relevance is critical for RAG apps
- Lots of passages in prompt → degraded quality

  → Can't only focus on recall
- Incorrect passages in prompt → possibly well-grounded yet wrong answers

  → Helps to establish thresholds for "good enough" grounding data



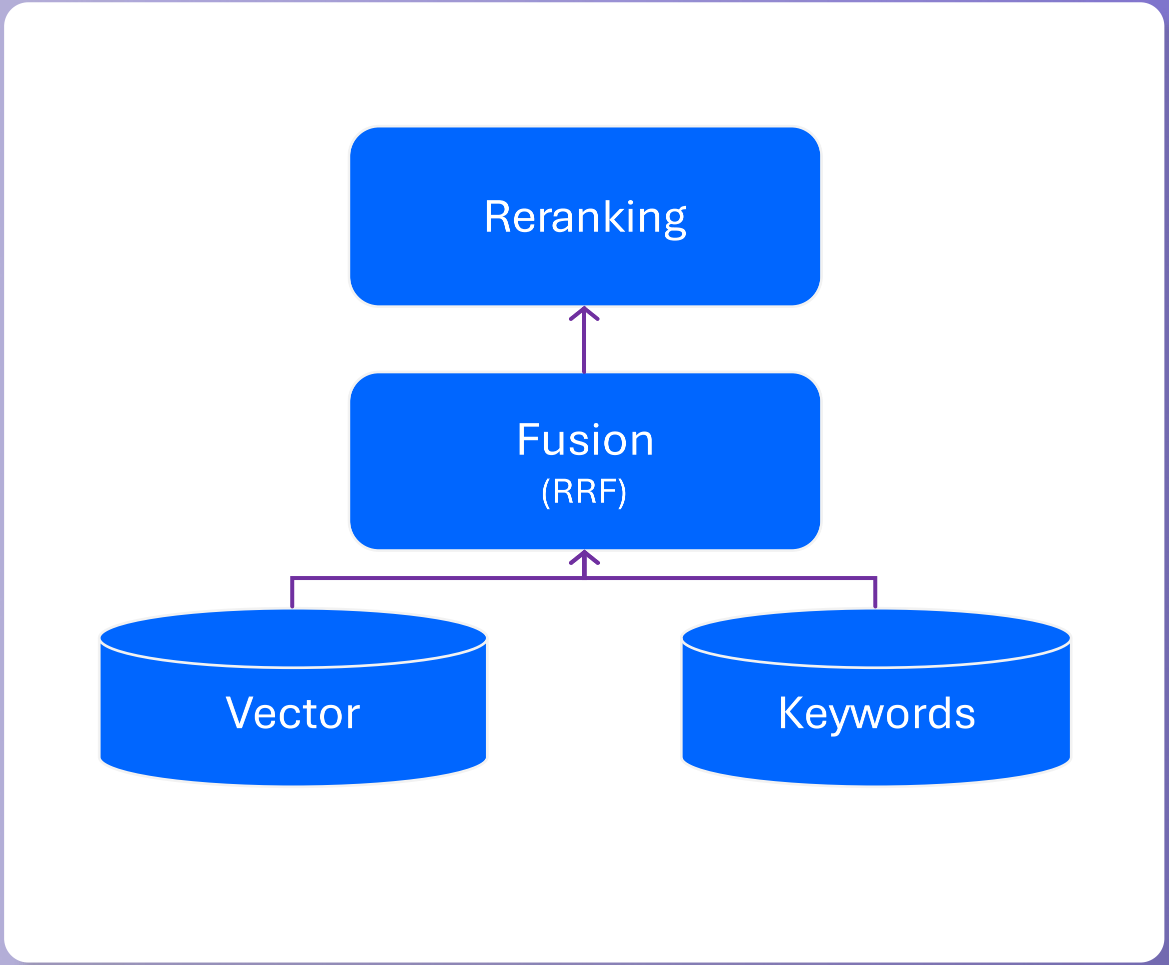
Source: Lost in the Middle: How Language Models Use Long Contexts, Liu et al. arXiv:2307.03172

Optimal retrieval in Azure AI Search

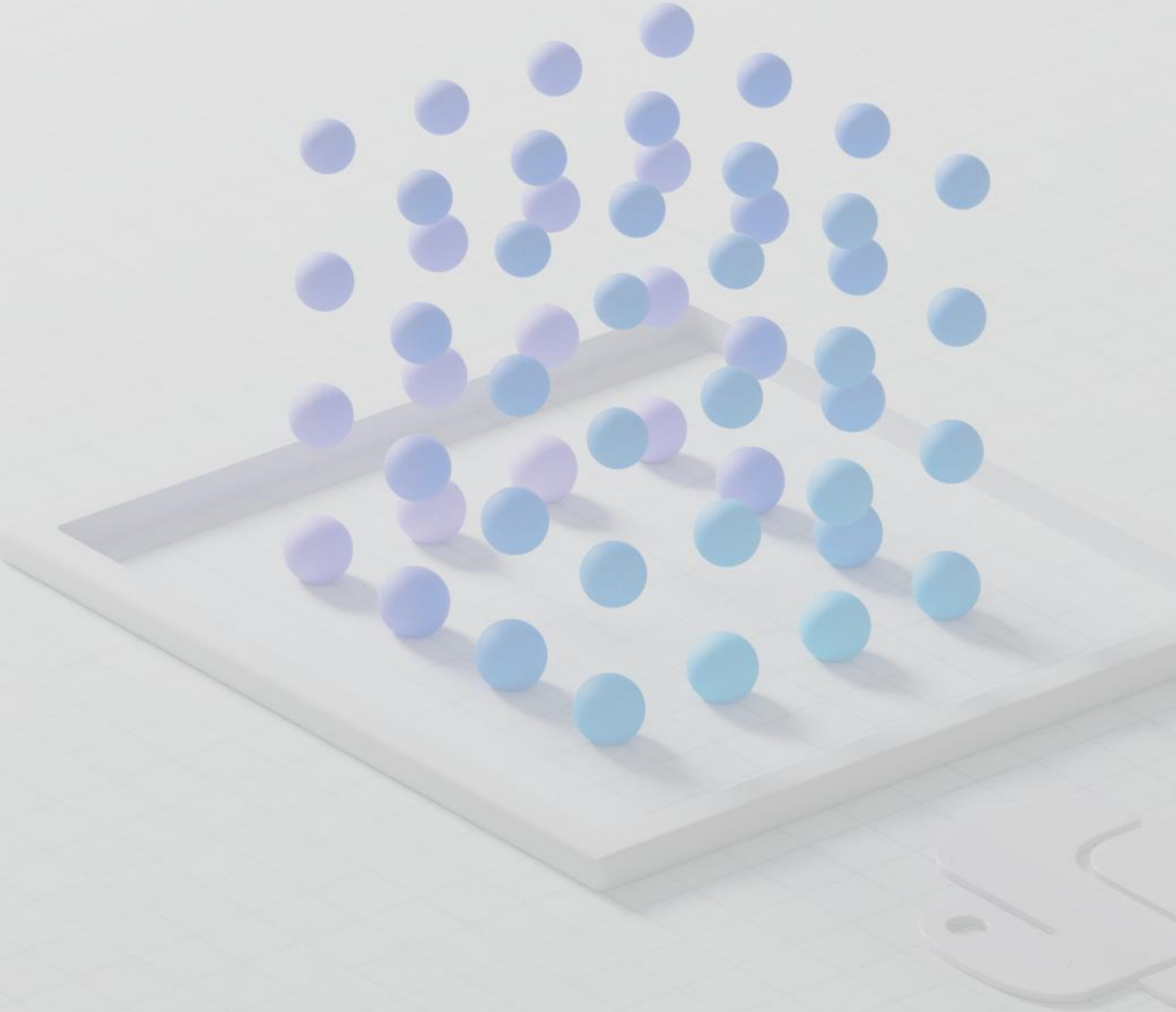
Complete search stacks do better:

Hybrid retrieval (keywords + vectors) > pure-vector or keyword

Hybrid + Reranking > Hybrid



Vector search



Vector embeddings

An embedding encodes an input as a list of floating-point numbers.

”dog” → [0.017198, -0.007493, -0.057982, 0.054051, -0.028336, 0.019245,…]

Different models output different embeddings, with varying lengths.

Model	Encodes	Vector length
word2vec	words	300
Sbert (Sentence-Transformers)	text (up to ~400 words)	768
OpenAI ada-002	text (up to 8191 tokens)	1536
Azure Computer Vision	image or text	1024

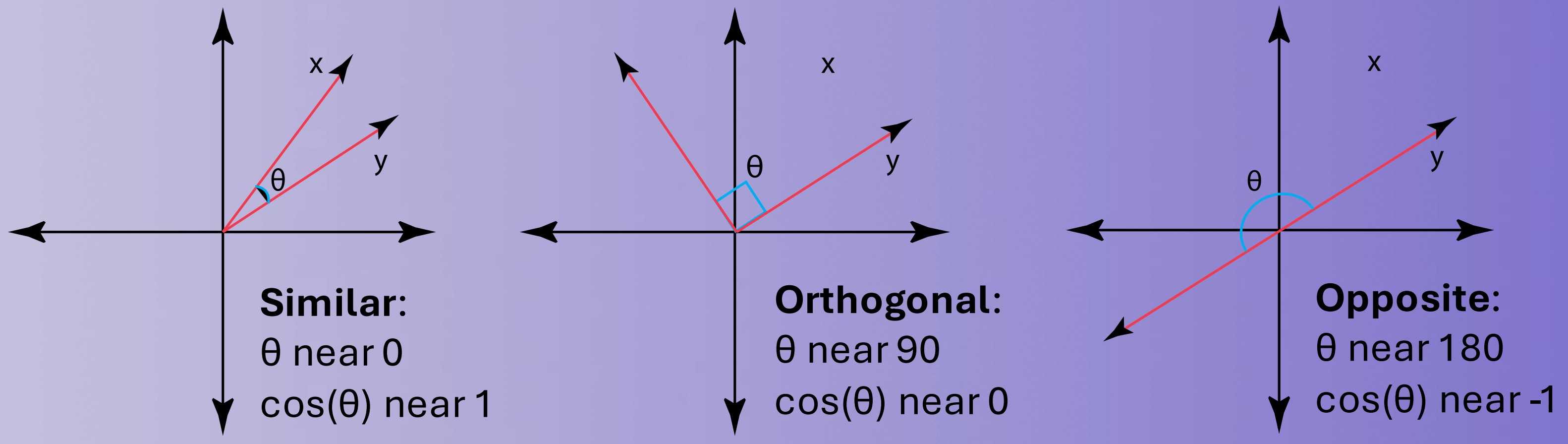
….and many more models!

🔗 Demo: Compute a vector with ada-002 (aka.ms/aitour/vectors)

Vector similarity

We compute embeddings so that we can calculate similarity between inputs. The most common distance measurement is **cosine similarity**.

```
def cosine_sim(a, b):
 return dot(a, b) /
 (mag(a) * mag(b))
```



Similar:
θ near 0
cos(θ) near 1

Orthogonal:
θ near 90
cos(θ) near 0

Opposite:
θ near 180
cos(θ) near -1

*For ada-002, cos(θ) values range from 0.7-1

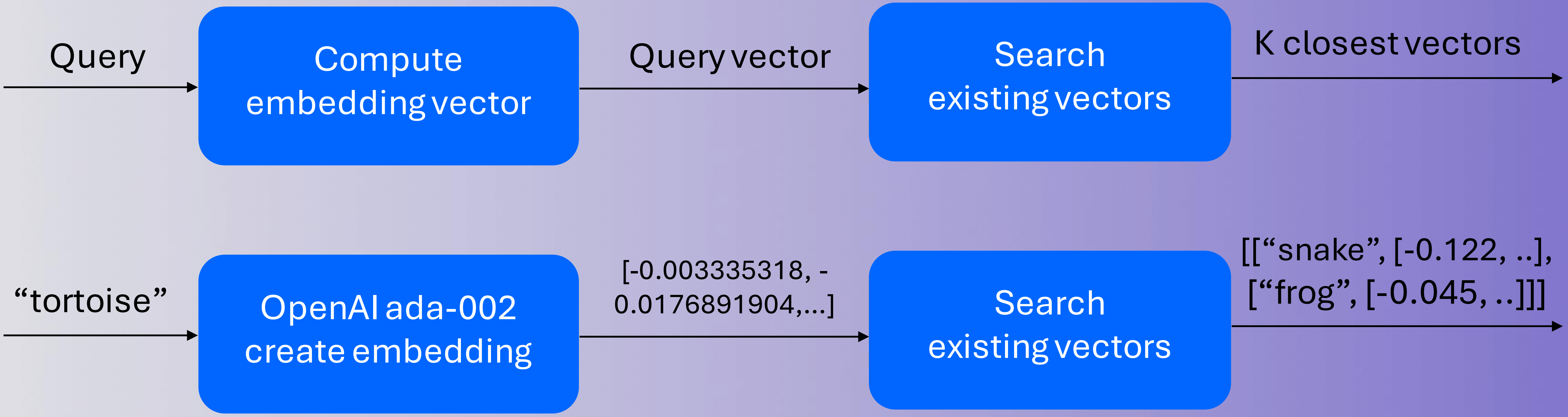
Demo: Compare vectors with cosine similarity (aka.ms/aitour/vectors)

Demo: Vector Embeddings Comparison (aka.ms/aitour/vector-similarity)

Vector search

1. Compute the embedding vector for the query
2. Find K closest vectors for the query vector

   Search exhaustively or using approximations



Query

Compute embedding vector

Query vector

Search existing vectors

K closest vectors

“tortoise”

OpenAI ada-002 create embedding

[-0.003335318, -0.0176891904,...]

Search existing vectors

[[“snake”, [-0.122, ..], [“frog”, [-0.045, ..]]]

💻 Demo: Search vectors with query vector (aka.ms/aitour/vectors)

Vector search in Azure AI Search

Generally available

Comprehensive vector search solution

Enterprise-ready

→ scalability, security and compliance

Integrated with Semantic Kernel, LangChain, LlamaIndex, Azure OpenAI Service, Azure AI Studio, and more

Demo: Azure AI search with vectors (aka.ms/aitour/azure-search)

Vector search strategies

ANN search

- ANN = Approximate Nearest Neighbors
- Fast vector search at scale
- Uses HNSW, a graph method with excellent performance-recall profile
- Fine control over index parameters

```
r = search_client.search(
        None,
        top=5,
        vector_queries=[VectorizedQuery(
            vector=search_vector,
            k_nearest_neighbors=5,
            fields="embedding")])
```

Exhaustive KNN search

- KNN = K Nearest Neighbors
- Per-query or built into schema
- Useful to create recall baselines
- Scenarios with highly selective filters
    - e.g., dense multi-tenant apps

```
r = search_client.search(
        None,
        top=5,
        vector_queries=[VectorizedQuery(
            vector=search_vector,
            k_nearest_neighbors=5,
            fields="embedding",
            exhaustive=True)])
```

Microsoft

Rich vector search query abilities

Filtered vector search

- Scope to date ranges, categories, geographic distances, access control groups, etc.
- Rich filter expressions
- Pre-/post-filtering
  - Pre-filter: great for selective filters, no recall disruption
  - Post-filter: better for low-selectivity filters, but watch for empty results

```
r = search_client.search(
        None,
        top=5,
        vector_queries=[VectorizedQuery(
            vector=query_vector,
            k_nearest_neighbors=5,
            fields="embedding")],
        vector_filter_mode=VectorFilterMode.PRE_FILTER,
        filter=
    "tag eq 'perks' and created gt 2023-11-15T00:00:00Z")
```

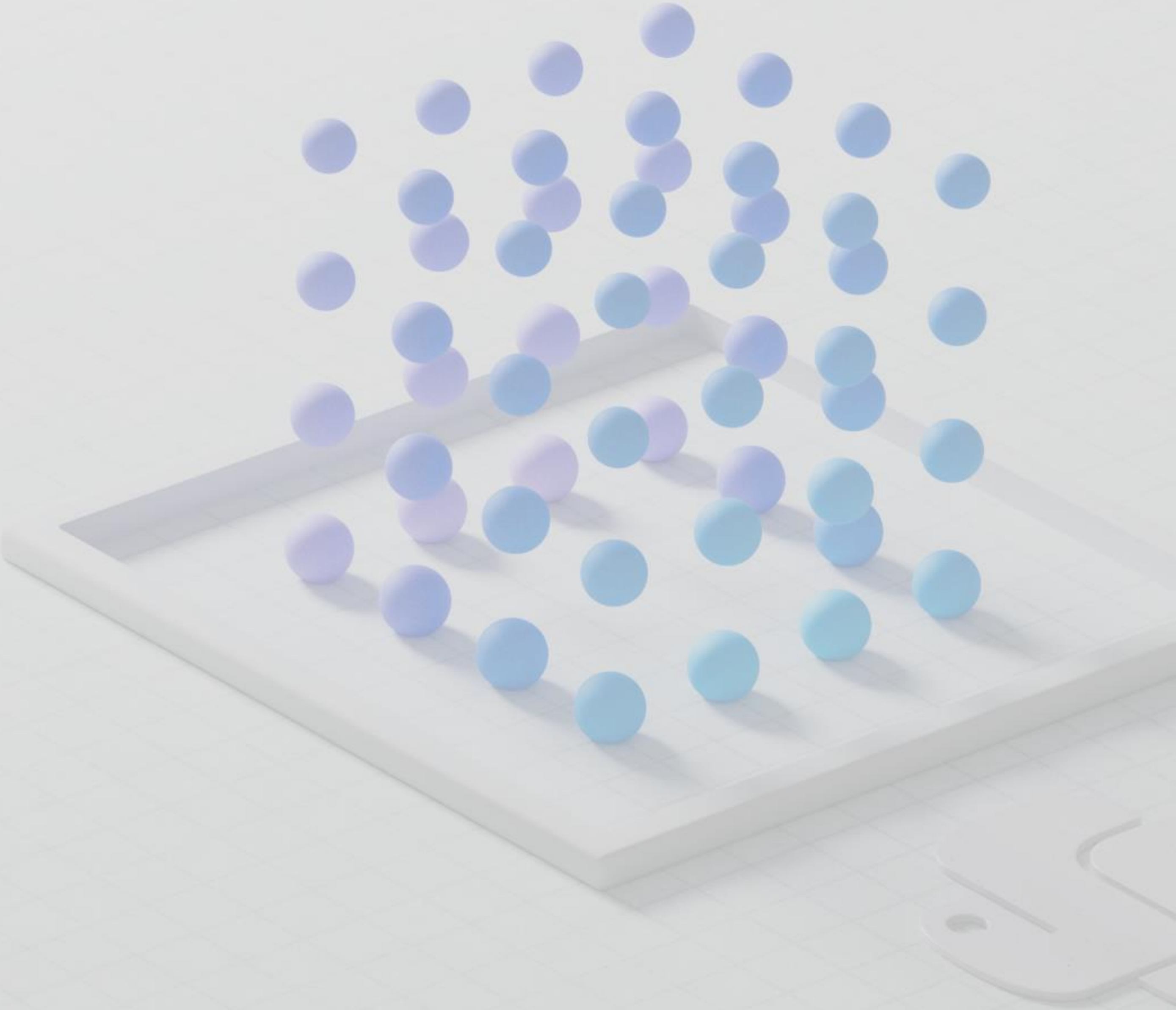
Filters in vector queries (aka.ms/aisearch/vectorfilters)

Multi-vector scenarios

- Multiple vector fields per document
- Multi-vector queries
- Can mix and match as needed

```
r = search_client.search(
        None,
        top=5,
        vector_queries=[
         VectorizedQuery(
             vector=query1, fields="body_vector",
             k_nearest_neighbors=5,),
         VectorizedQuery(
             vector=query2, fields="title_vector",
             k_nearest_neighbors=5,)
        ])
```

Hybrid search



Microsoft

Optimal retrieval in Azure AI Search

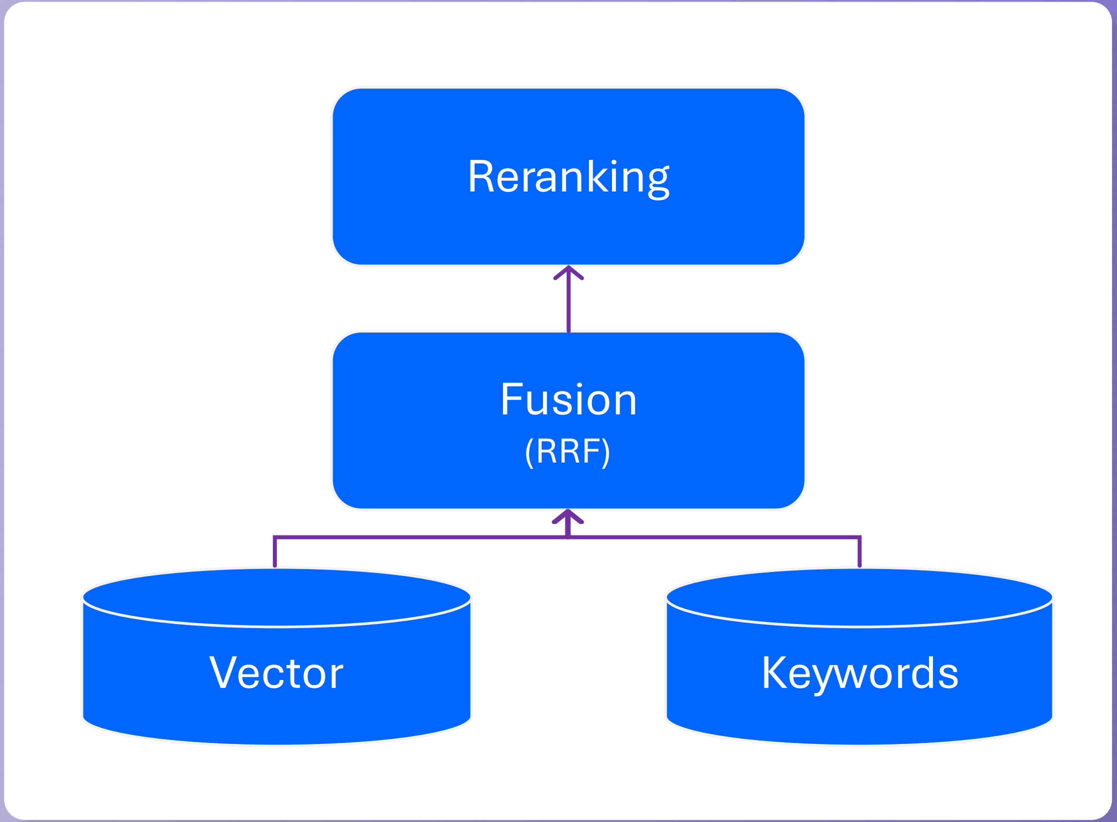
Complete search stacks do better:

Hybrid retrieval (keywords + vectors) > pure-vector or keyword

Hybrid + Reranking > Hybrid

Identify good & bad candidates

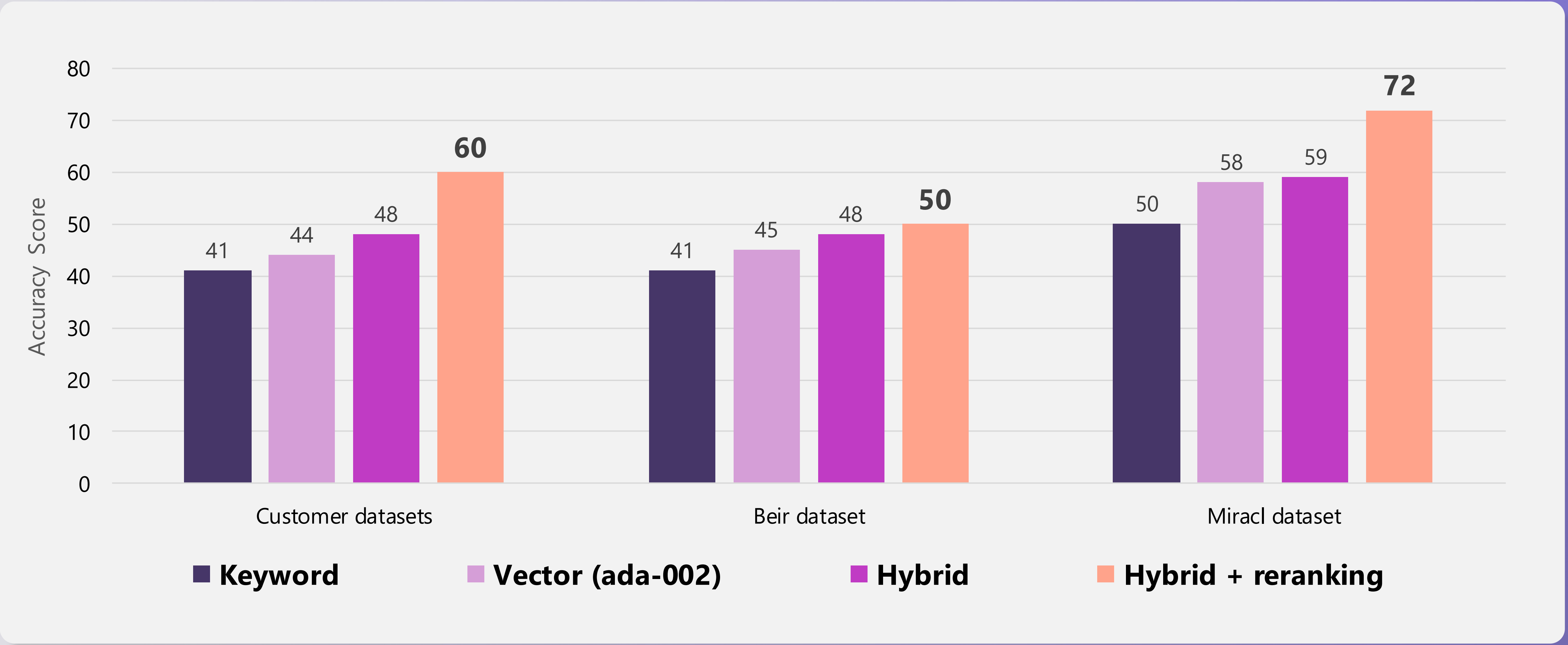
Normalized scores from semantic ranker
Exclude documents below a threshold



Demo: Compare text, vector, hybrid, reranker (aka.ms/aitour/search-relevance)

Microsoft

Retrieval relevance across methods



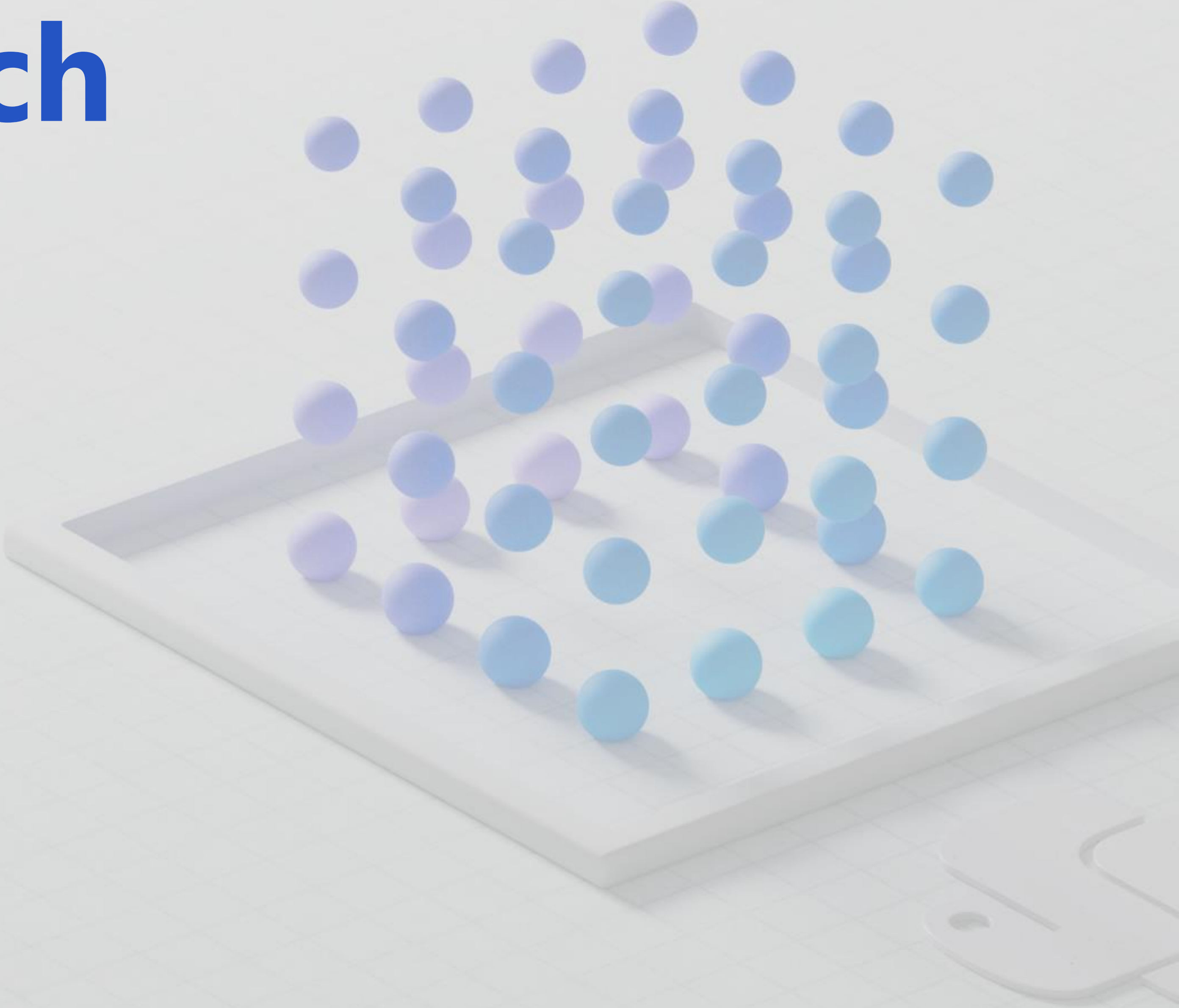
Outperforming vector search with hybrid + reranking (aka.ms/ragrelevance)

Impact of query types on relevance

Query type	Keyword [NDCG@3]	Vector [NDCG@3]	Hybrid [NDCG@3]	Hybrid + Semantic ranker [NDCG@3]
Concept seeking queries	39	45.8	46.3	59.6
Fact seeking queries	37.8	49	49.1	63.4
Exact snippet search	51.1	41.5	51	60.8
Web search-like queries	41.8	46.3	50	58.9
Keyword queries	79.2	11.7	61	66.9
Low query/doc term overlap	23	36.1	35.9	49.1
Queries with misspellings	28.8	39.1	40.6	54.6
Long queries	42.7	41.6	48.1	59.4
Medium queries	38.1	44.7	46.7	59.9
Short queries	53.1	38.8	53	63.9

Outperforming vector search with hybrid + reranking (aka.ms/ragrelevance)

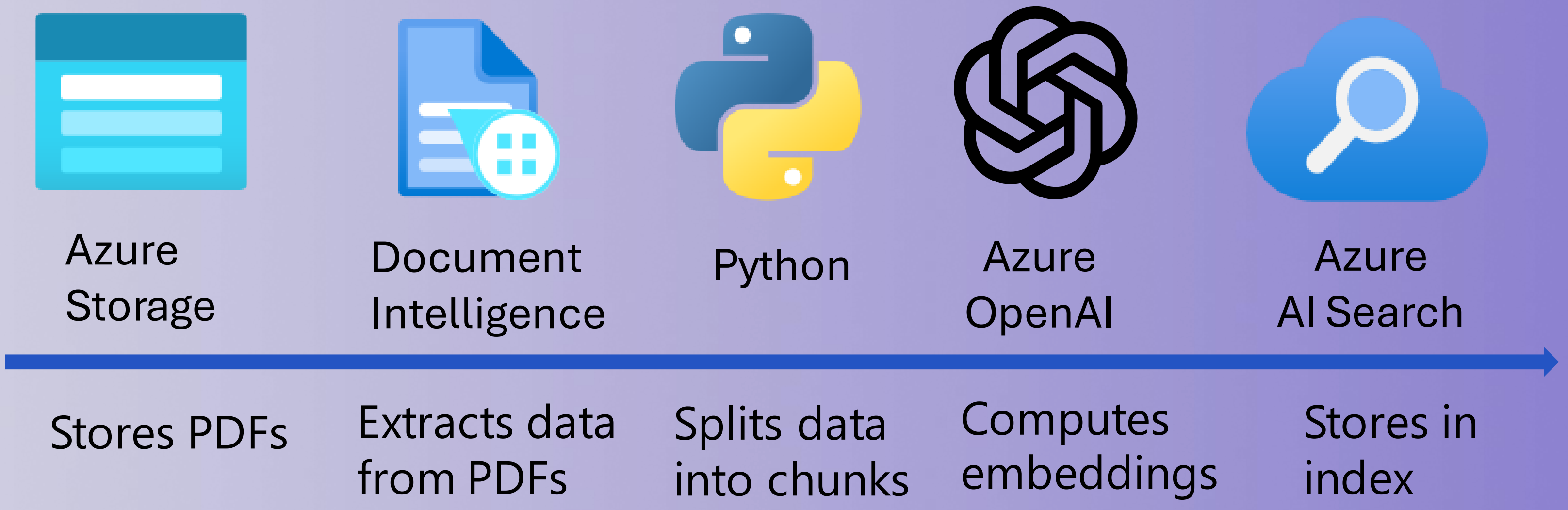
Azure AI search data indexing



Manual indexing

You can use the SDK to write your own code to add data to an index.

Example:
prepdocs.py

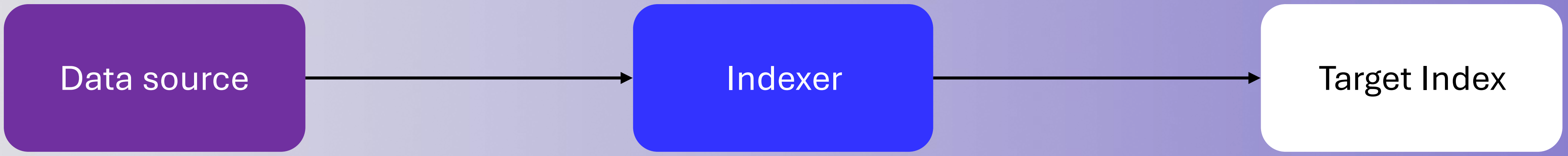


Azure Storage	Stores PDFs
Document Intelligence	Extracts data from PDFs
Python	Splits data into chunks
Azure OpenAI	Computes embeddings
Azure AI Search	Stores in index

Data ingestion guide: Adding documents   aka.ms/ragchat/add-data

Cloud-based indexing

Indexers: Connect the search service to a cloud data source, and it will index the data periodically or on a trigger.



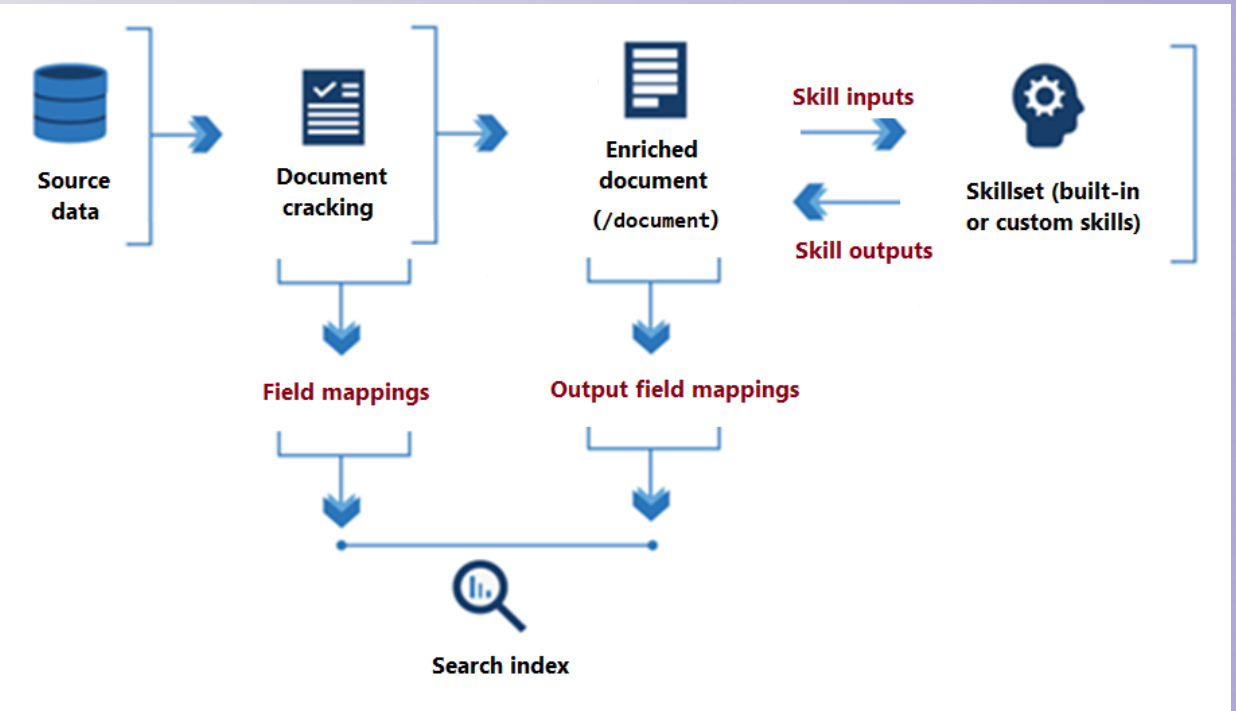
- Azure Blob Storage
- Azure Cosmos DB
- Azure Data Lake Storage Gen2
- Azure SQL Database
- SharePoint in Microsoft 365
- Azure Cosmos DB for MongoDB

...and more!

Indexers in Azure AI Search (aka.ms/aisearch/indexers)

Skillsets for indexers

Skillset: A set of skills that prepare a document for indexing, calling either built-in AI search functions or custom code.

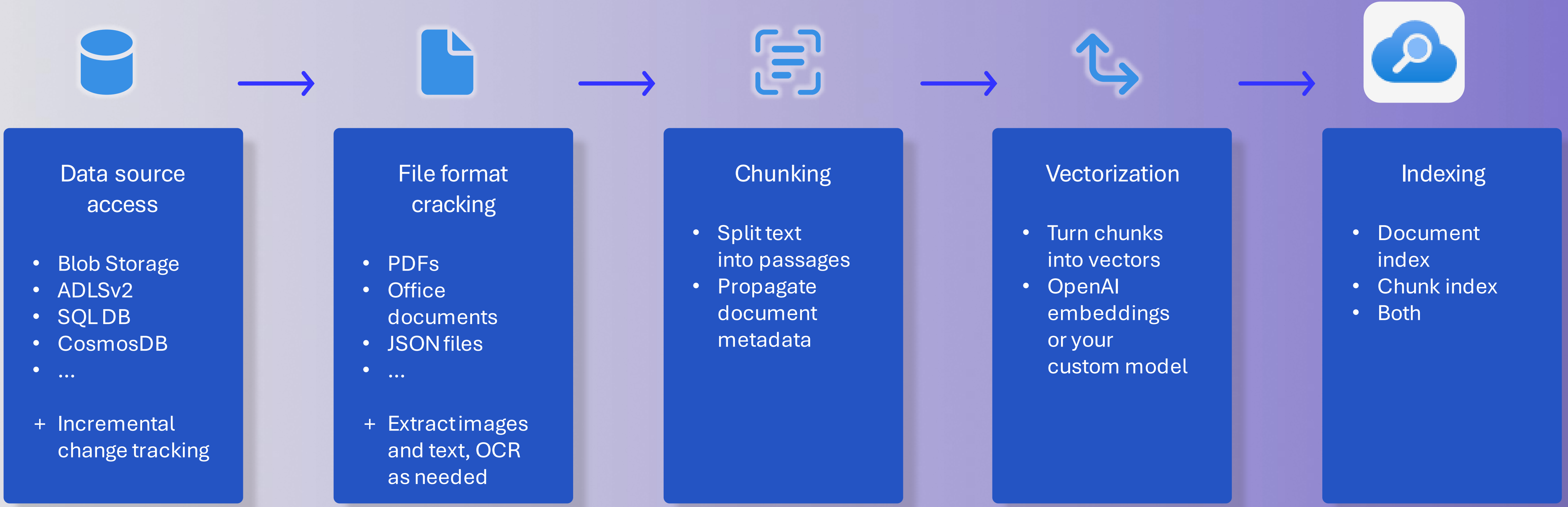


Skillset concepts in Azure AI Search (aka.ms/aisearch/skillsets)

Integrated vectorization

In preview

A combination of indexers and built-in skills for chunking and vectorization.



Data source access

- Blob Storage
- ADLSv2
- SQL DB
- CosmosDB
- …

+ Incremental change tracking

File format cracking

- PDFs
- Office documents
- JSON files
- …

+ Extract images and text, OCR as needed

Chunking

- Split text into passages
- Propagate document metadata

Vectorization

- Turn chunks into vectors
- OpenAI embeddings or your custom model

Indexing

- Document index
- Chunk index
- Both

Integrated data chunking and embedding in Azure AI Search (aka.ms/integrated-vectorization)

Integrated vectorization in RAG chat repo

Once the PR is merged, you can opt to use it via:

```
azd env set USE_FEATURE_INT_VECTORIZATION true
azd up
```

🔗 PR: Adding integrated vectorization support (aka.ms/ragchat/intvect)

Manual indexing vs. Integrated vectorization

Pros:

- All code is local and easy to change.

Cons:

- Hard to connect to indexers for cloud-based data.
- Has to be manually re-run for new data.

Pros:

- Easily connect to indexers that can add new data on triggers or periodically.
- You don’t need to maintain chunking or embedding code yourself.

Cons:

- Currently in preview mode.
- Customizing the skills takes more effort, if the built-in skills are not sufficient.

Azure AI search advanced features

Analyzers

Analyzers are components of the full-text search engine for processing strings during indexing and query execution.

- **Language analyzers**: If you’re indexing non-English documents in particular, consider customizing the analyzer used.
- **Custom analyzers**: Useful for custom tokenization, like to recognize phone numbers, word normalization, etc.

🔗 Analyzers for text processing in Azure AI Search (aka.ms/aisearch/analyzers)

Scoring profiles

Scoring profiles are criteria for boosting a search score based on custom parameters.

```
"scoringProfiles": [
  {
   "name": "boostKeywords",
   "text": {
     "weights": {
       "HotelName": 2,
       "Description": 5 }
   }
  }
]
```

🔗 Add scoring profiles to boost search scores (aka.ms/aisearch/scoring)

Next steps

- Register for the hackathon → aka.ms/hacktogether/chatapp

- Introduce yourself in our discussion forum
- Deploy the repo with the sample data
  - See steps on low cost deployment → aka.ms/ragchat/free

- Start customizing the project!
- Post in forum if you have any issues deploying or questions about customization. 🙋🏼‍♀️ 🙋🏾‍♀️ 🙋🏻‍♀️ 🙋🏼‍♀️ 🙋🏿

- Join tomorrow’s session: GPT-4 with Vision