MCW TECHNOLOGIES

WORKSHOP

Hitchhiker's Guide to
Building an AI Solution Now



Brian A. Randell

brianr@mcwtech.com

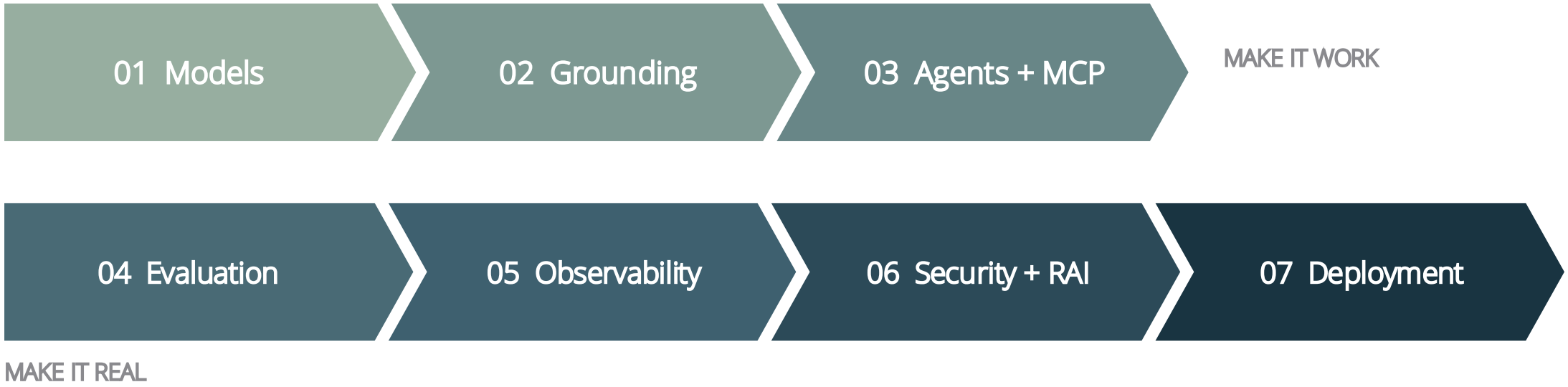


Matt Nunn

mattnunn@mcwtech.com

Seven patterns, two halves

Not one big app. Each pattern on its own, in code you can take home.



Throughout: what it costs, what you trade, and where each pattern fits. Including what to skip.

The sample apps

Three apps and a handful of single-file demos

Chat bot

Document Q&A with citations.

Azure AI Search, Azure OpenAI, C#.

Patterns 02, 04, 05, 06, 07

ConferencePulse

Live polls and Q&A for a session, on the composable .NET AI stack.

IChatClient, DataIngestion, VectorData, Qdrant, MCP.

Patterns 01, 02, 03

SQL hybrid RAG

The database decides: write a SQL query or search by meaning.

Azure SQL vector type, stored procedure.

Pattern 02

Single-file demos

One C# file per idea. Models, tools, structured output.

Patterns 01, 03

Where I started

Credit where it is due

App	Started from	By	What I changed
Chat bot	Azure-Samples/ azure-search-openai-demo-csharp	Microsoft	Lots – see repo docs
ConferencePulse	luisquintanilla/ dotnet-ai-conference-assistant	Luis Quintanilla, .NET Blog (MSFT PM)	Real deployment to Azure Entra support Security – see docs
SQL hybrid RAG	Azure-Samples/ azure-sql-db-ai-samples-search	Davide Mauri (former MSFT)	New C# + SQL version

The originals are good starting points. We made large changes with GitHub Copilot and Claude Code to the two bigger samples, including security hardening, so these are published as my own versions with links back, not as forks.

Get the code

https://github.com/brianrandell/live360mco26



AI foundations

What a useful AI solution needs

What changes in our work

The work we value	Craft, judgment and experience The worry about losing them
The work we delegate	Drafting and repetitive tasks A first pass we can inspect
The work we own	Choose the problem Decide whether it worked

Audience prompt: which task would you gladly hand over?

What AI contributes

1	**Draft and explain**	Explore an approach Produce something to inspect
2	**Find and organise**	Work with supplied evidence Prepare records and documents
3	**Act through tools**	Run a bounded task Check the actual result

A useful output still needs a purpose, evidence and a check

A prompt with a clear job



One task and a clear audience

Separate unrelated requests

Supply the relevant evidence

State scope and constraints

Specify the output you need

Define how to check it

A useful answer starts with a clear question

A demo and a dependable answer

The first demo

Ten sample sessions
One question with a known answer

The full event

Hundreds of sessions
Room changes and cancellations

Acceptance checks	Does it use the current schedule? Can attendees open its sources?

A conference assistant works on ten sessions. Will it handle the full event?

An empty result needs an explanation

The tempting answer

“Nothing is scheduled”
Based on an empty result

Other possible causes

The schedule feed failed
A filter excluded every match

An explicit result

Separate empty from unavailable
Check source and query status

The app returns no sessions. That does not explain why.

The boundary of a change



Clarify the search label

Keep search behaviour intact

Ranking also changes

Relevant sessions disappear

Approve ranking changes separately

Retest the same search queries

A search-label edit should not silently change which sessions appear

The developer's judgment

1

Problem

Understand the person and the job

2

Brief

Set scope and constraints

3

Review

Inspect evidence and changes

4

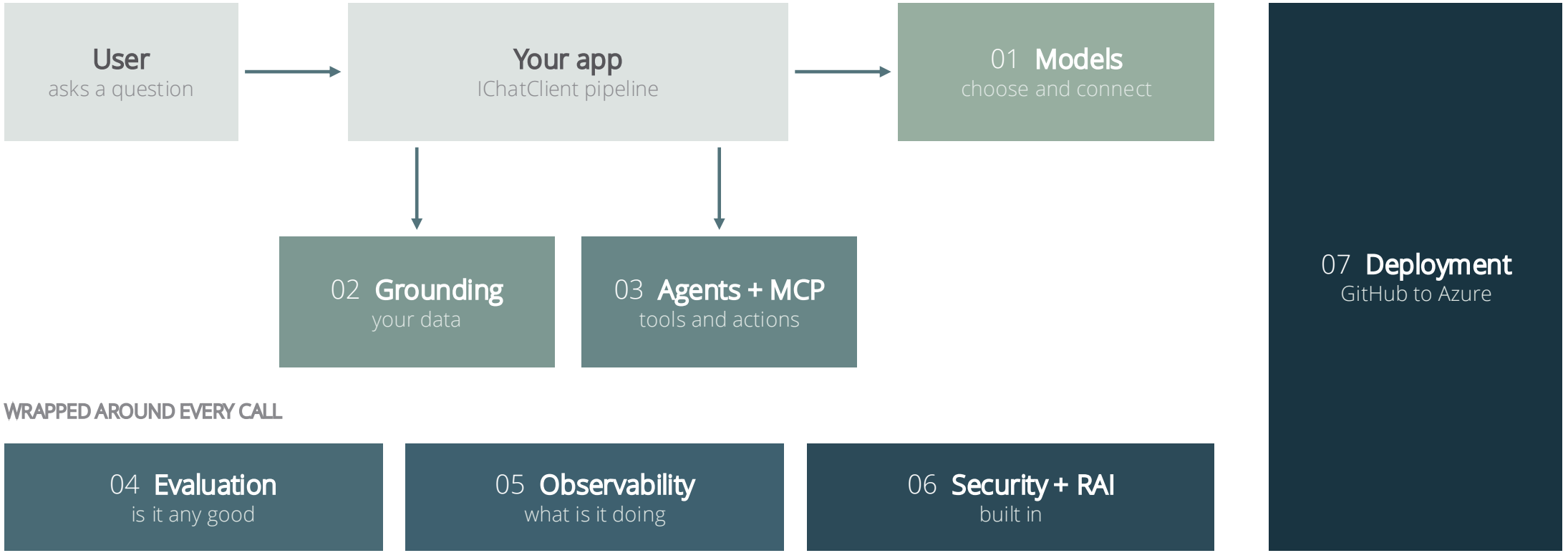
Acceptance

Try the result as a user

AI can do the legwork while you guide the work

Anatomy of an AI solution

The map we will keep coming back to



Seven patterns. One map. We light up one piece at a time.

Which model fits the job?

A smaller model

Try it on a bounded task
Check accuracy and response time

A frontier model

Test demanding reasoning
Compare the extra cost and delay

The useful comparison

Same task, evidence and checks
Choose on accepted results

Model size is a choice to test, not a guarantee of quality

Three places in one solution

1. **The interface**
   Where you chat
   Where history is stored

2. **The model**
   Where inference runs
   Which provider receives context

3. **The tools and data**
   Where actions execute
   Which files and services they use

A local interface can still call a cloud model

One AI workflow across your devices

Desktop
Little Rascal

MacBook
Dramatic Exit

Z Fold 8
Flexible Demeanour

ChatGPT account and project
Conversation history and supplied context

Shared files
OneDrive: latest deck and talk track
Versioned handoff for the next session

Working session
Tools, permissions and execution
Depend on the active environment

Chat continuity, file sync and session continuity need separate checks

Where your AI work actually lives



ChatGPT cloud

Conversation history and project context

Check account, project, branch and mode

Codex workspace

Local machine or cloud execution

Check repo, paths, runtime and permissions

Explicit file access and save-back

OneDrive · canonical working files

Slide decks and talk tracks · synced Mac copies · phone access

Our case: Z Fold 8 → Mac → branch into Work mode

The conversation followed us. File editing needed a different mode here.

A focused prompt for ConferencePulse

Too broad	Explain the workshop, plan my app and compare every AI provider
One clear task	Find where chunking is explained Use the supplied session material
An answer we can check	Give a short answer and citation Say when the evidence is missing

Then ask the next question as a separate, reviewable step

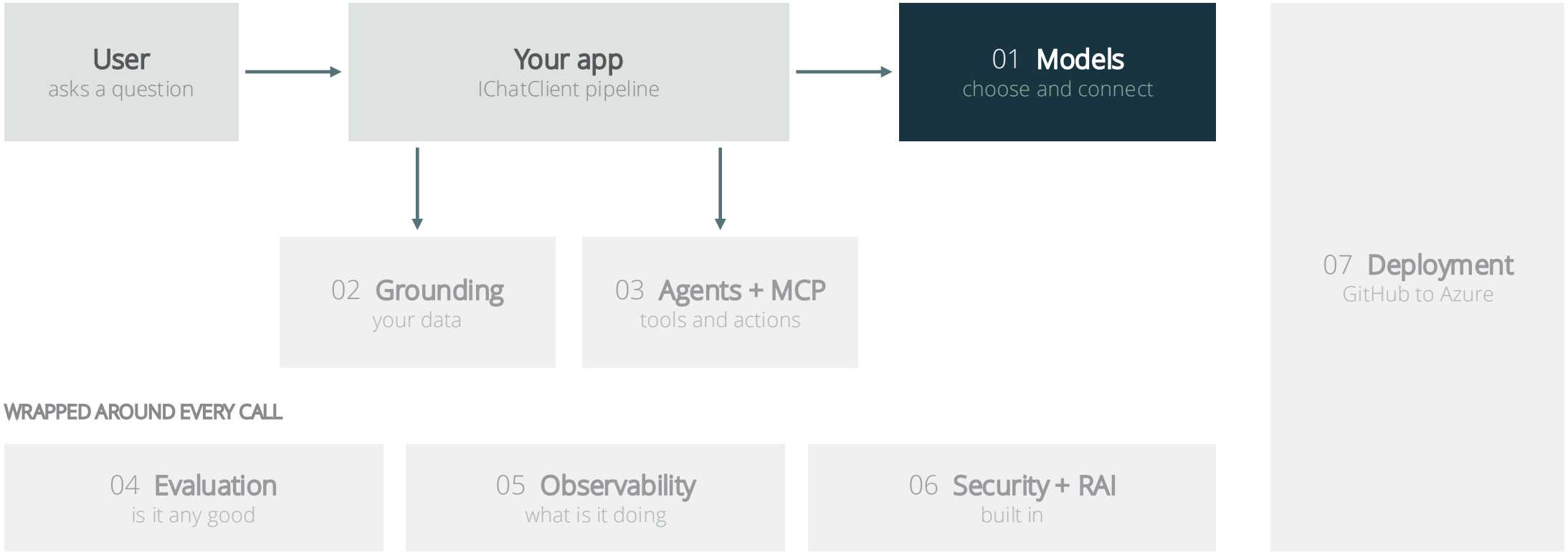


Models

Choosing them and connecting to them

Models

Pattern 01



Which model, running where, reached how?

Choosing a model provider

The "Now" in the title, demonstrated

ORIGINAL TOOLSET

Technologies: GitHub Models, Microsoft Foundry, Microsoft Agent Framework, MCP, GitHub, Azure.

WHAT CHANGED

GitHub Models was retired on July 30, 2026.

Playground, catalog, and inference API are gone. GitHub points application workloads to Microsoft Foundry.

The lesson is the pattern for this whole section:

code against an abstraction, not a provider.

Source: GitHub Changelog, July 1 and July 30, 2026; Microsoft Learn migration guide.

Picking a model

Six questions before you look at a leaderboard

Does it pass your tests?

Your evaluation set decides, not a benchmark.

How much context?

What has to fit in one call: instructions, history, retrieved evidence, tool results.

How fast?

Time to first token and total time. A chat UI and a batch job want different things.

What does a call cost?

Input and output tokens are priced separately. Reasoning tokens count.

Where can the data go?

Region, tenant, and contract. Sometimes the answer is: nowhere.

What else can it do?

Tool calling, structured output, images, audio. Check before you design around it.

Where models run

	Cloud, your tenant	Self-hosted	Local device
Example	Microsoft Foundry model deployments	Open-weight models on your own GPUs	Foundry Local, Ollama
You get	Largest models, managed scale, content filters, keyless auth	Full control of data and versions	Offline, private, no per-call bill
You pay	Per token, or reserved capacity	Hardware, power, and people	Developer hardware
Watch for	Region availability, quotas, model retirements	Capacity planning, patching, model quality	Smaller models, hardware variety
Fits when	Most production apps	Hard data-boundary or sovereignty rules	Inner dev loop, edge, demos

Same code for all three if the app talks to an abstraction. Thursday: AIH04, Local-First AI Development.

One abstraction: IChatClient

Microsoft.Extensions.AI

```
using Microsoft.Extensions.AI;

// Foundry, Azure OpenAI, Ollama, Foundry Local…
IChatClient client = GetClientFor(provider);

ChatResponse response =
    await client.GetResponseAsync("What is AI?");

Console.WriteLine(response);
```

IChatClient

Chat, streaming, tool calls

IEmbeddingGenerator

Text in, vectors out

Provider packages

Each one implements the same interfaces

Swap the provider and this code does not change. That is the whole trick.

The pipeline

Cross-cutting behavior as middleware, like ASP.NET Core




```
builder.Services.AddChatClient(services =>
    client.AsBuilder()
        .UseDistributedCache()      // do not pay twice for the same question
        .UseFunctionInvocation()    // let the model call your .NET methods
        .UseOpenTelemetry()         // traces and token counts, pattern 05
        .Build(services));
```

Three patterns appear in this pipeline. Write your own with DelegatingChatClient.

DEMO D1 Models

Same prompt, swap the provider

Single-file demos

SHOW

1. One file, one prompt, cloud model in Foundry
2. Change one line: local model
3. Streaming response
4. Structured output into a C# record

Fallback Pre-recorded run of each file. Local model already pulled.

Beyond a chat reply

Streaming

`GetStreamingResponseAsync`

Tokens as they arrive. Users forgive slow if they can see progress.

Structured output

`GetResponseAsync<T>`

Ask for a C# type, not a paragraph to parse.

Tool calling

`AIFunctionFactory.Create`

Describe a .NET method to the model and let it ask you to run it.

Embeddings

`IEmbeddingGenerator`

Turn text into vectors. The on-ramp to grounding.

Tool calling is where agents start. Embeddings are where grounding starts.

Models

Cost, tradeoffs, and where it fits

Lever	What you gain	What it costs you
Smaller model for easy calls	Lower cost and latency on most traffic	Routing logic, and two models to evaluate
Response caching	Repeat questions cost nothing	Stale answers, cache keys, privacy of cached content
Shorter prompts and context	Cheaper, faster, often more accurate	Work to decide what to leave out
Reserved capacity	Predictable latency and spend at volume	You pay whether you use it or not
Local model in the dev loop	No per-call bill while iterating	Behavior differs from the production model

Skip until you need it: fine-tuning, multi-model routing, and anything you cannot measure yet.

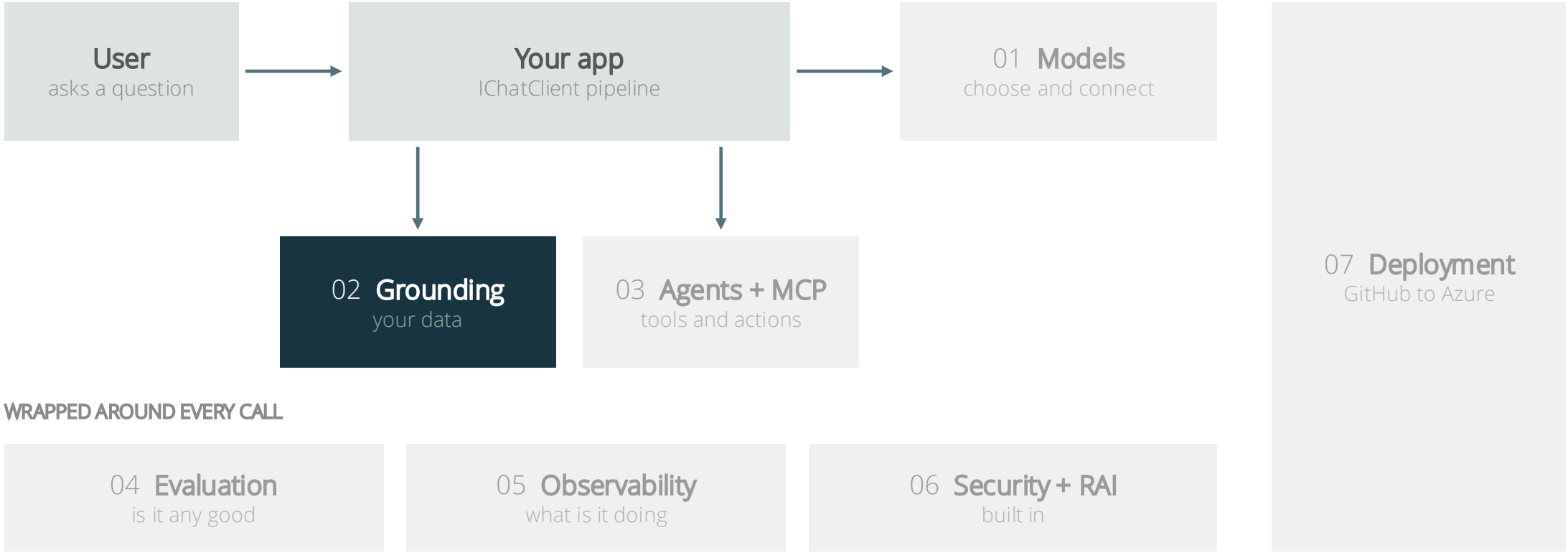


Data Readiness

Getting your Data Ready for AI

Grounding

Pattern 02



How does the model answer from your data, and show its evidence?

Business decisions need business data

Public context

Market and industry information
Useful external reference

Your business evidence

Private records and definitions
Current, permitted source material

An answer for your business

Use the relevant evidence
Show sources, scope and limits

Build company context into the application

The six-stage lifecycle



Ingest	Transform	Store	Describe	Consume	Test
Capture what arrived	Prepare identity and meaning	Publish a dependable copy	Attach scope and definitions	Retrieve permitted evidence	Verify the result and its limits

Ingest and Transform | **Store and Describe** | **Consume and Test**

People decide, the system does, and a check proves readiness

Ingest: what actually arrived?

The inputs	Session Markdown Polls, Q&A and other live events
The pitfall	A delivery is missing or late Absence looks like a real result
The check	Keep source IDs and timestamps Reconcile arrivals and replay safely

An empty result is not proof that nothing happened

Transform: preserve the meaning

Source context

“Cancelled” appears in a heading
The description still looks useful

Prepared records

Store cancellation as a field
Keep status with every chunk

ConferencePulse design

Filter recommendations by status
Retain the source and its version

A session marked “cancelled” must not become a recommendation

Store: keep a reproducible copy



Retain the source

Keep a stable reference

Chunks, vectors and metadata

Stable IDs for replacement

Refresh changed content

Remove superseded or deleted data

An index is a prepared copy with a lifecycle

Describe: what does the record mean?

1. **Time definition**
   Which Tuesday? Which time zone?
   Event time or attendee time?

2. **Session metadata**
   Session ID and event date
   Time zone, version and access

3. **The check**
   A travelling attendee sees the time
   With its time zone clearly stated

“Tuesday at nine” needs a date and a time zone

Consume: choose the evidence path

Passages

Retrieve relevant approved text
Cite the source and version

Numbers

Query a defined measure
Use the agreed scope and period

Permissions	Authorise before retrieval Apply access rules to derived copies

Finding a passage and calculating a total are different jobs

Test the answer and its boundaries

1

Known result

Match approved evidence

2

Missing data

Expose the gap

3

Ambiguity

Ask for clarification

4

Denied access

Reveal no restricted evidence

Repeat these cases when sources, indexes, rules or models change



Grounding

Answers from your data, with evidence

One question through all six stages



Where was chunking explained?

Which session and source apply?

Capture, prepare and index

Keep identity, scope and version

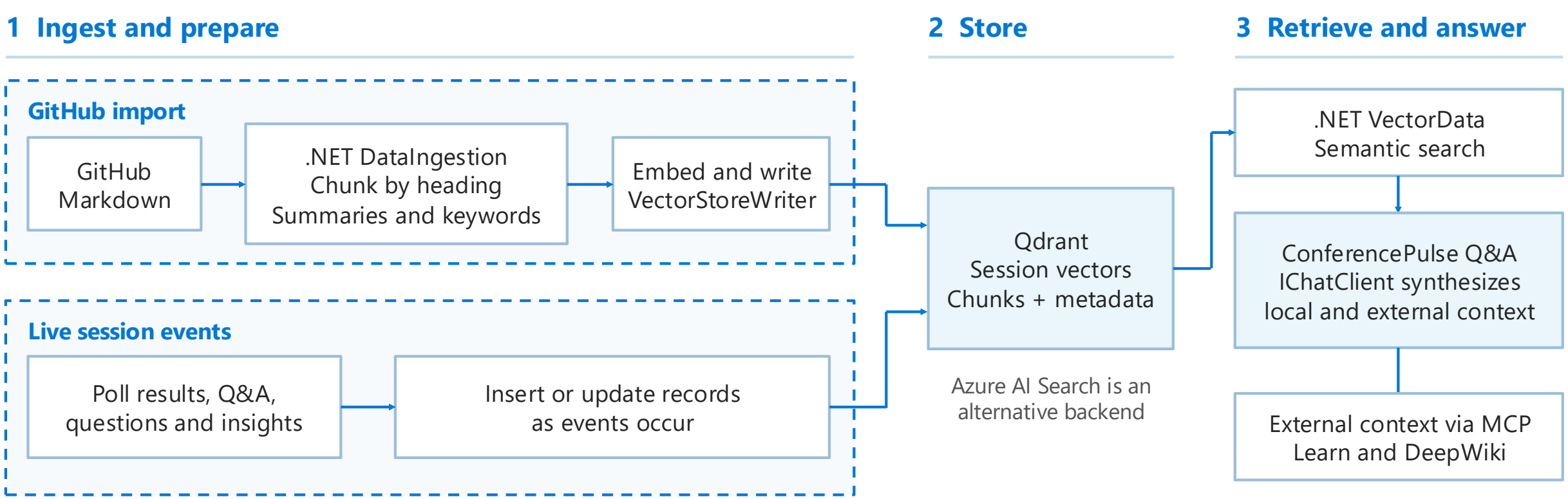
Retrieve the right passage

Open the citation and check it

Use the architectures to locate the work we just discussed

ConferencePulse on .NET with Qdrant

Imported session material and live events feed the same searchable knowledge base

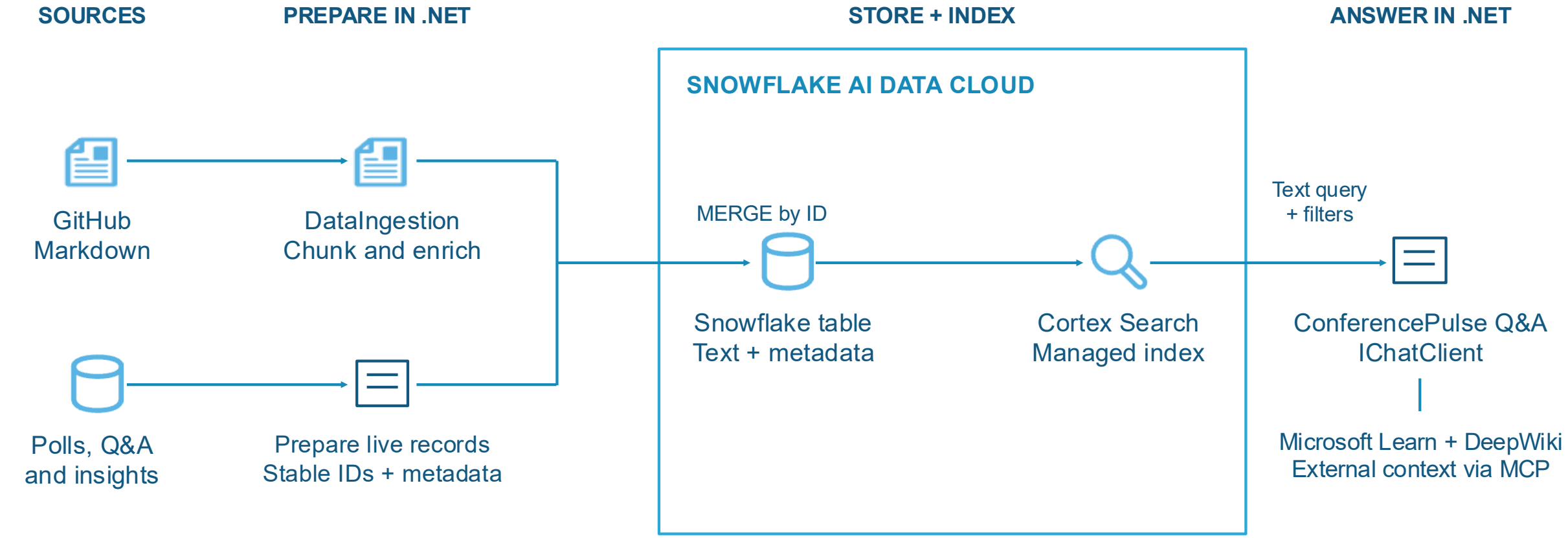


TEST + OPERATE Check retrieved chunks, live-event freshness and answer grounding.

HUMAN DECISIONS Choose repositories, chunking settings, enrichment and vector backend.

ConferencePulse with Snowflake

Proposed backend using a Snowflake table and managed Cortex Search

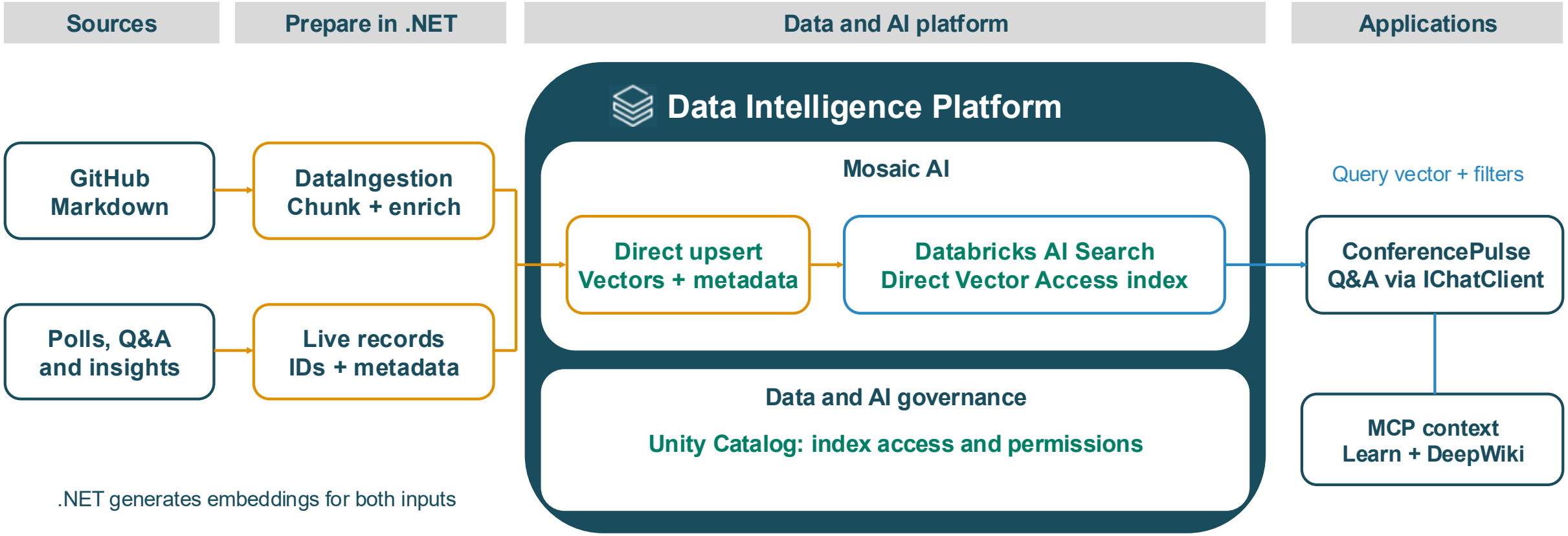


APP CHANGE Write text to Snowflake and query Cortex Search through an adapter.

DEMO CHECK Verify source IDs, search freshness and retrieval scope.

ConferencePulse with Databricks

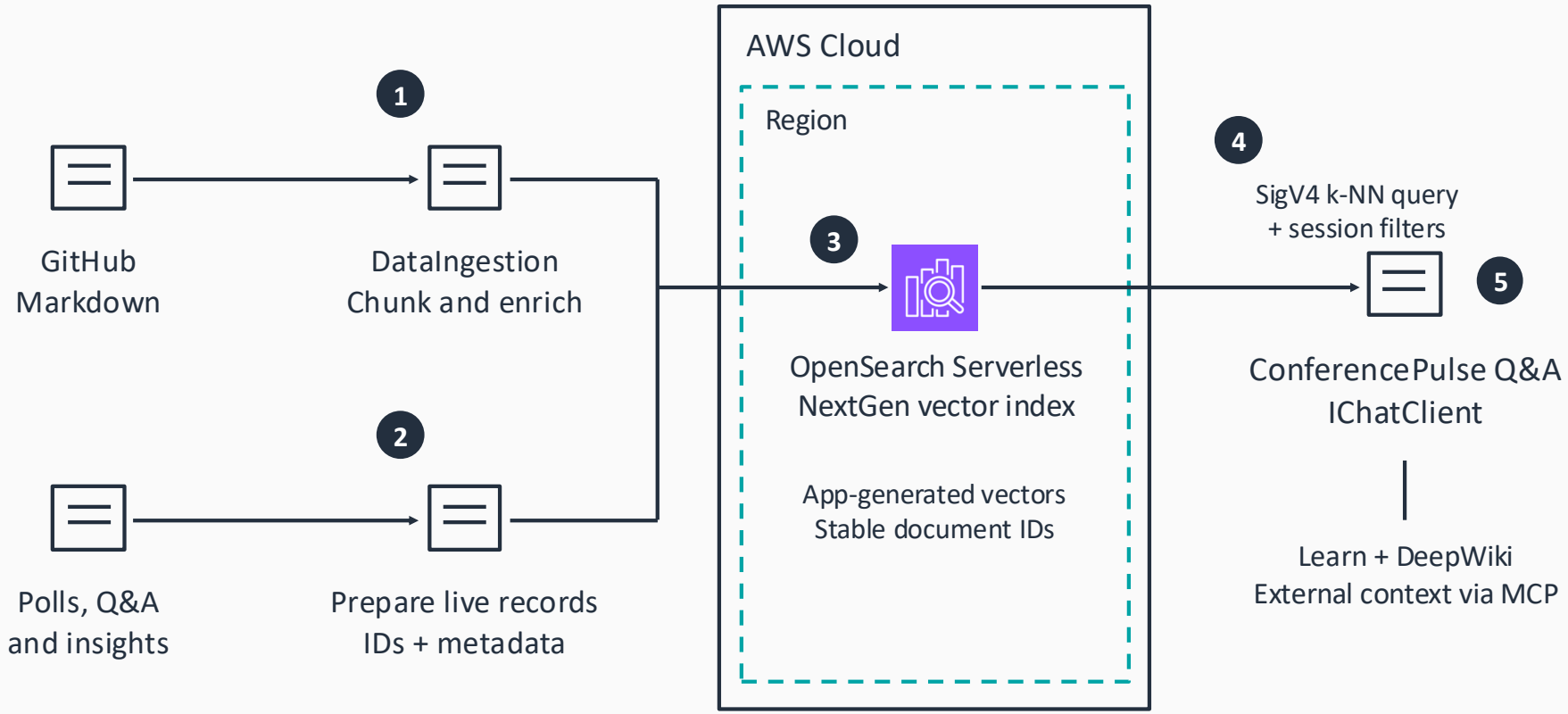
Proposed backend using a Direct Vector Access index



APP CHANGE	Use a Databricks REST adapter for direct upserts and vector queries.
DEMO CHECK	Check index readiness, vector dimensions and session filters.

ConferencePulse with AWS

Proposed backend using an OpenSearch Serverless vector index



App adapter signs indexing and search calls with SigV4.

1 Prepare GitHub content

Read, chunk and enrich Markdown with DataIngestion.

2 Prepare live events

Keep stable IDs and session metadata.

3 Embed and index

Write vectors to a NextGen collection using document IDs.

4 Search for evidence

Run k-NN with session filters. Retrieve content and metadata.

5 Answer in ConferencePulse

Combine local evidence and MCP context with IChatClient.

6 Verify the backend

Check replacement by ID, freshness and IAM access.

RAG in one picture

Retrieval-augmented generation: prepare once, answer many times

PREPARE · when content changes



ANSWER · on every question



Most wrong answers are retrieval problems, not model problems. If the right chunk never reaches the prompt, no model can save you.

Six stages of AI-ready data

Matt Nunn's six-stage model



Ingest	Transform	Store	Describe	Consume	Test
Get the raw data. Know what arrived and what is missing.	Make records work together. Make hidden meaning explicit.	A home for each prepared copy. A version you can reproduce.	Business meaning: definitions, owners, metadata.	A controlled path to the evidence, within the caller's permissions.	Known answers, and agreed behavior at the edges.

Each stage: people decide, the system does, and there is a check that says you can hand it on.

Data preparation is most of the work in a real RAG system.

Two kinds of questions, two paths

Structured path

Counts, totals, trends

The model writes a query against a schema you describe.

The database computes the answer. The model explains it.

"What share of attendees chose vector search?"

Document path

Policies, notes, session material

Search finds the passages closest in meaning.

The model answers from those passages and cites them.

"Where did the speaker explain chunking?"

Real apps need both, and something has to choose. Use the caller's permissions on both paths.

DEMO D2

Grounding

Two questions, one database

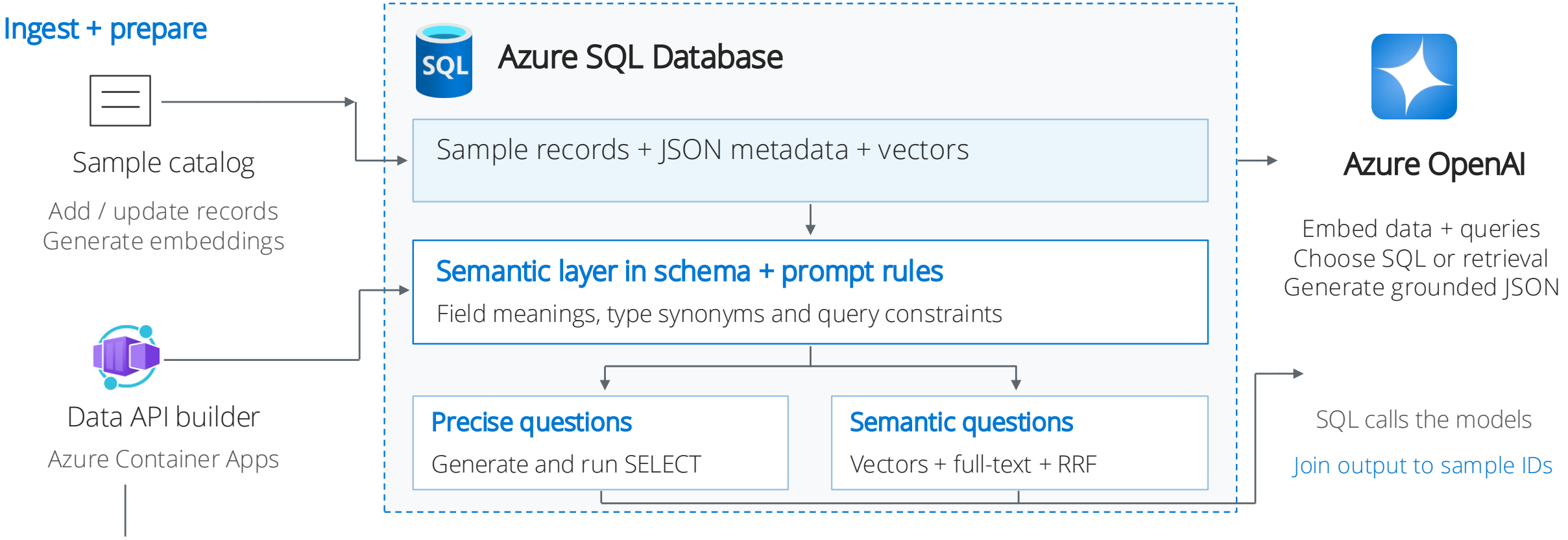
SQL hybrid RAG · Davide's sample

SHOW

1. Ask the counting question. Show the SQL it wrote.
2. Ask the meaning question. Show the vector search.
3. Open the stored procedure that chooses
4. The structured JSON that comes back

Hybrid RAG on Microsoft Azure SQL

SQL answers precise questions. Semantic retrieval finds relevant samples.

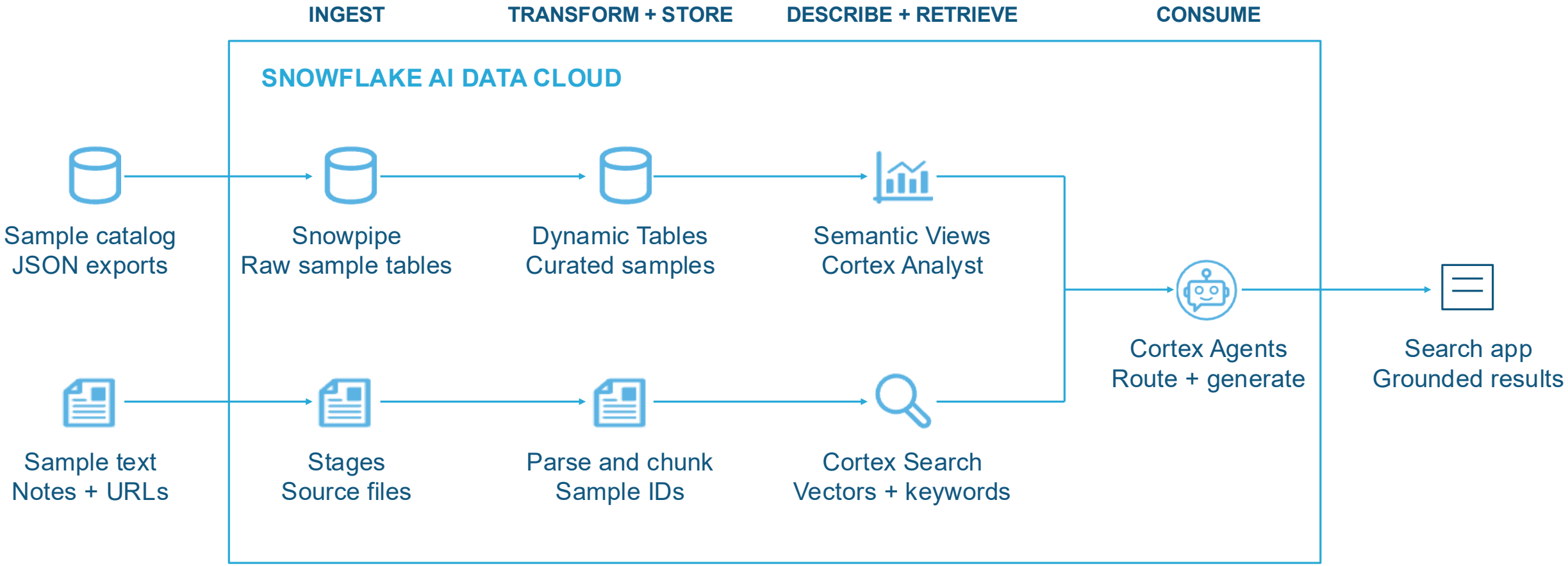


HYBRID RAG

SQL or retrieval, then generation. Retrieval merges vector and keyword rankings (R

Hybrid RAG with Snowflake

Proposed samples-search adaptation: structured catalog + descriptive text

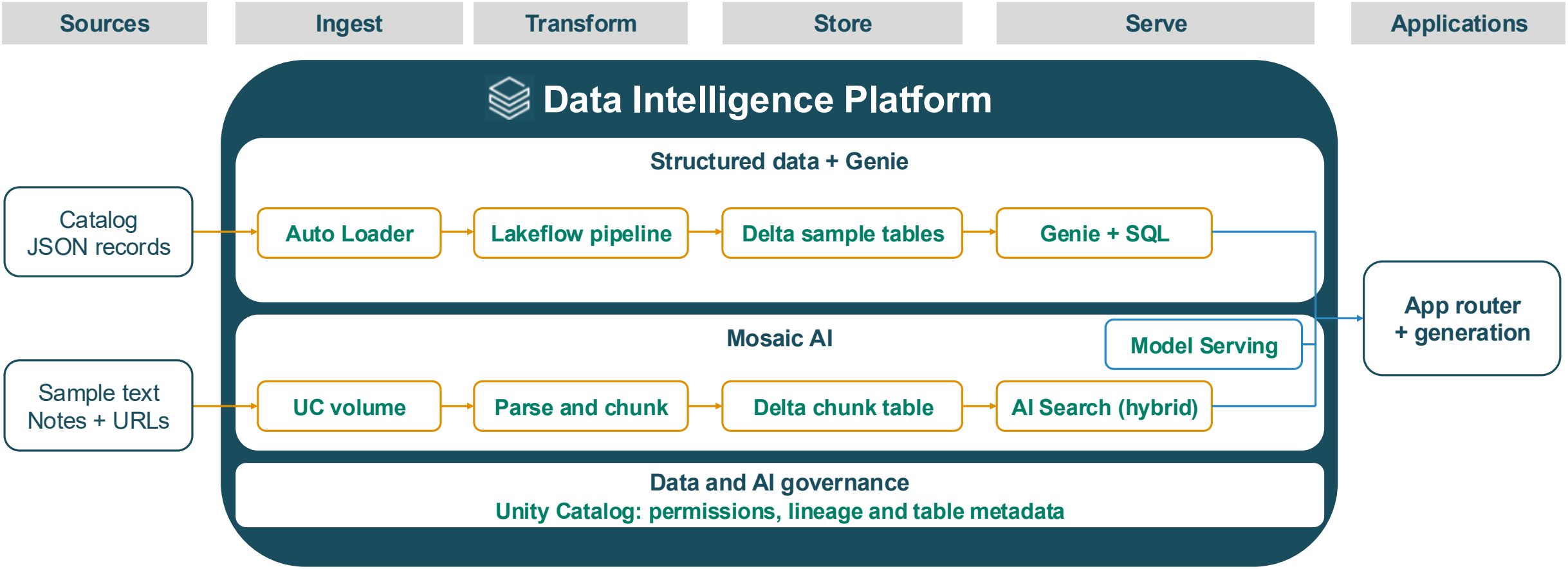


HYBRID RAG Cortex Agents uses SQL, hybrid search, or both to compose an answer.

SEMANTIC LAYER Semantic views define field meanings, synonyms and relationships for SQL.

Hybrid RAG with Databricks

Proposed samples-search adaptation: Genie SQL + AI Search behind an app router

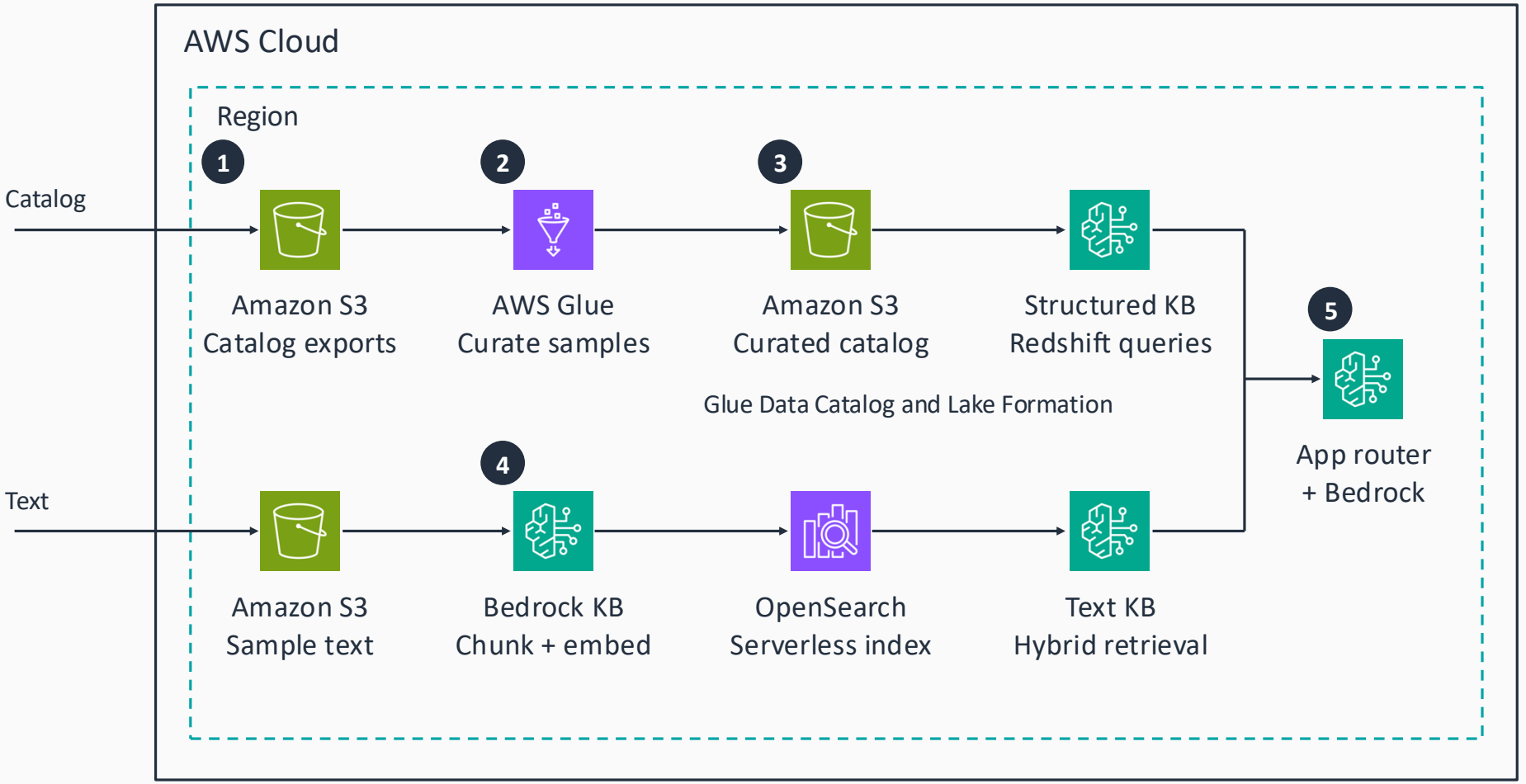


HYBRID RAG App router uses Genie SQL, AI Search (vectors + keywords), or both.

SEMANTIC LAYER Genie knowledge store: field meanings, synonyms, joins and SQL examples.

Hybrid RAG with AWS

Proposed samples-search adaptation using Amazon Bedrock Knowledge Bases



1 Ingest

Land catalog exports and descriptive text in S3.

2 Transform

Normalize sample fields.
Keep stable IDs and URLs.

3 Semantic layer

Table and column meanings.
Curated SQL examples.

4 Hybrid retrieval

Bedrock KB + OpenSearch.
Vector and keyword search.

5 Hybrid RAG

App routes SQL or search.
Bedrock composes answers.

6 App changes

Replace SQL procedures

Semantic layer: Bedrock query context and curated SQL examples.

Where vectors live

In your search service

Azure AI Search

Keyword, vector, and hybrid search with ranking. Document-level access control.

Pick when: search quality is the product.

In your database

Azure SQL, SQL Server 2025

A vector type next to the rows you already have. One place to secure and back up.

Pick when: your data and team already live in SQL.

In a vector store

Qdrant and others

Purpose-built, runs anywhere including a container on your laptop.

Pick when: you want portability or run outside Azure.

Microsoft.Extensions.VectorData: one interface over many stores, so this choice is not a rewrite.

DEMO D3

Grounding in code

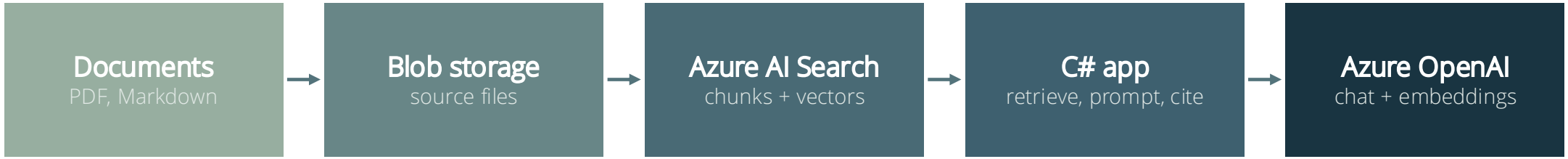
Ask the documents, get citations

Chat bot · Azure AI Search + Azure OpenAI

SHOW

1. Ask a question the documents answer
2. Open the citation and check it
3. Ask something the documents do not cover
4. Look at the retrieval call and the prompt

Chat bot: how it fits together



WHERE IT STARTED

Azure-Samples/azure-search-openai-demo-csharp

A complete reference app from Microsoft. Good bones. Not recently invested in.

WHAT I CHANGED

Too Much!

See APP-CHANGES.MD in the sample

We come back to this app for evaluation, observability, and security.

The composable version

Microsoft.Extensions.DataIngestion (preview)



MarkdownReader	Image alt text	HeaderChunker, semantic chunker	SummaryEnricher, KeywordEnricher	VectorStoreWriter

```
using IngestionPipeline<string> pipeline =
    new(reader, chunker, writer)
{
    ChunkProcessors = { summaryEnricher }
};

await foreach (var result in pipeline.ProcessAsync(dir))
    Console.WriteLine(result.DocumentId);
```

Every stage is swappable.

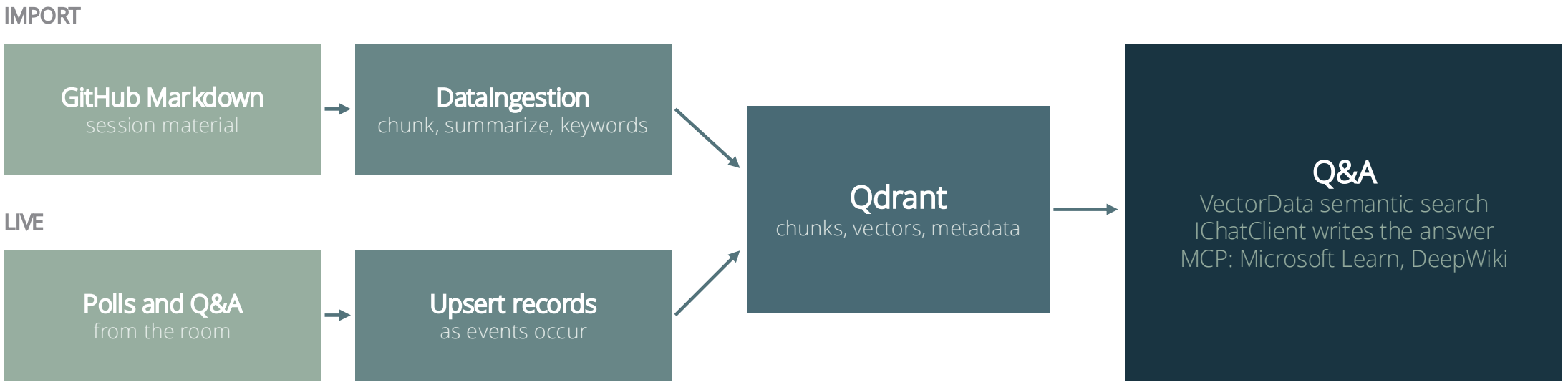
Enrichers call the model through IChatClient.

The writer embeds and stores through VectorData.

Preview package. Expect changes.

ConferencePulse

Session material and live events feed one knowledge base



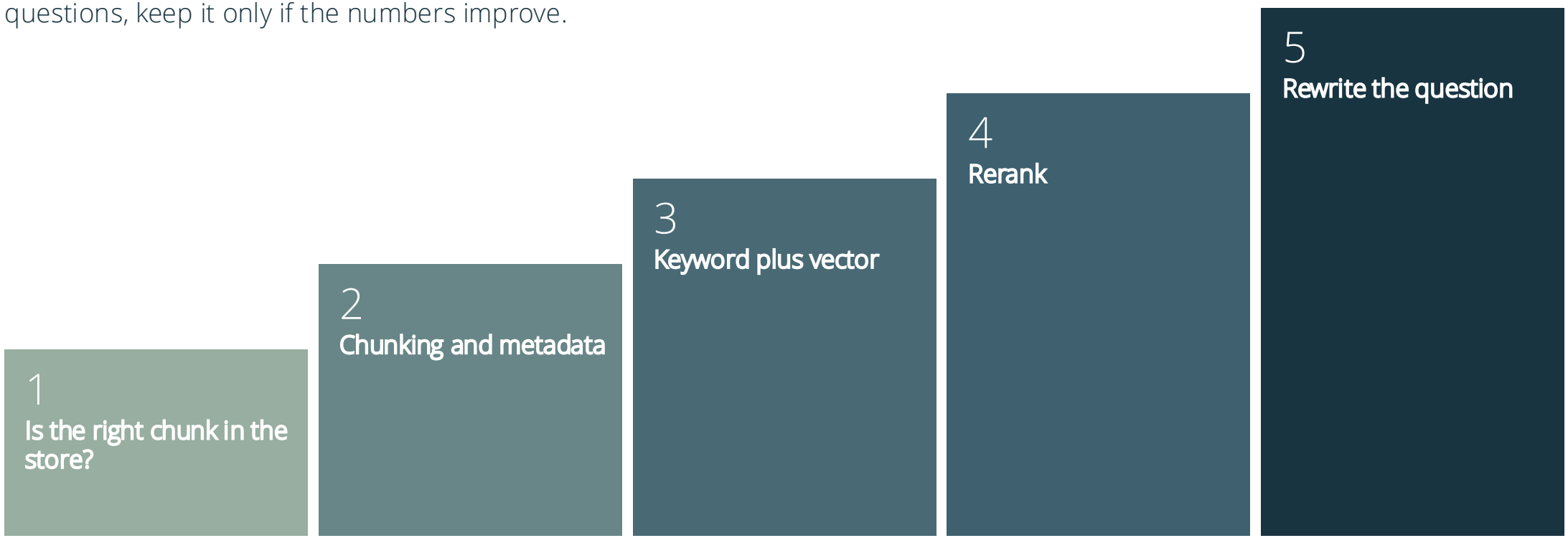
Started from: Luis Quintanilla's conference assistant, published on the .NET Blog.

Tons of Changes. See APP-CHANGES.MD in the sample. My version replaces Qdrant in the cloud in pgvector, PostgreSQL extension

When the answer is wrong

Work the retrieval levers in this order

Cheapest fix first. Change one thing, rerun your test questions, keep it only if the numbers improve.



Search for it directly. Half the time it was never ingested.

Split on structure, keep headings, attach source and date.

Exact terms and meaning together beat either alone.

Fetch more, then let a ranker pick the best few.

Turn chat history into a standalone search query.

Grounding

Cost, tradeoffs, and where it fits

Lever	What you gain	What it costs you
Smaller chunks	More precise matches	Lost context, more chunks to store and search
Enrichment at ingest	Better recall from summaries and keywords	A model call per chunk, every time content changes
More evidence per answer	Fewer misses	More input tokens on every question
Managed search service	Ranking, filters, access control built in	Another service on the bill, tied to one cloud
Vectors in your database	One system to secure and operate	Fewer search features than a search engine

Skip until you need it: fine-tuning for knowledge, graph RAG, and a second vector store.

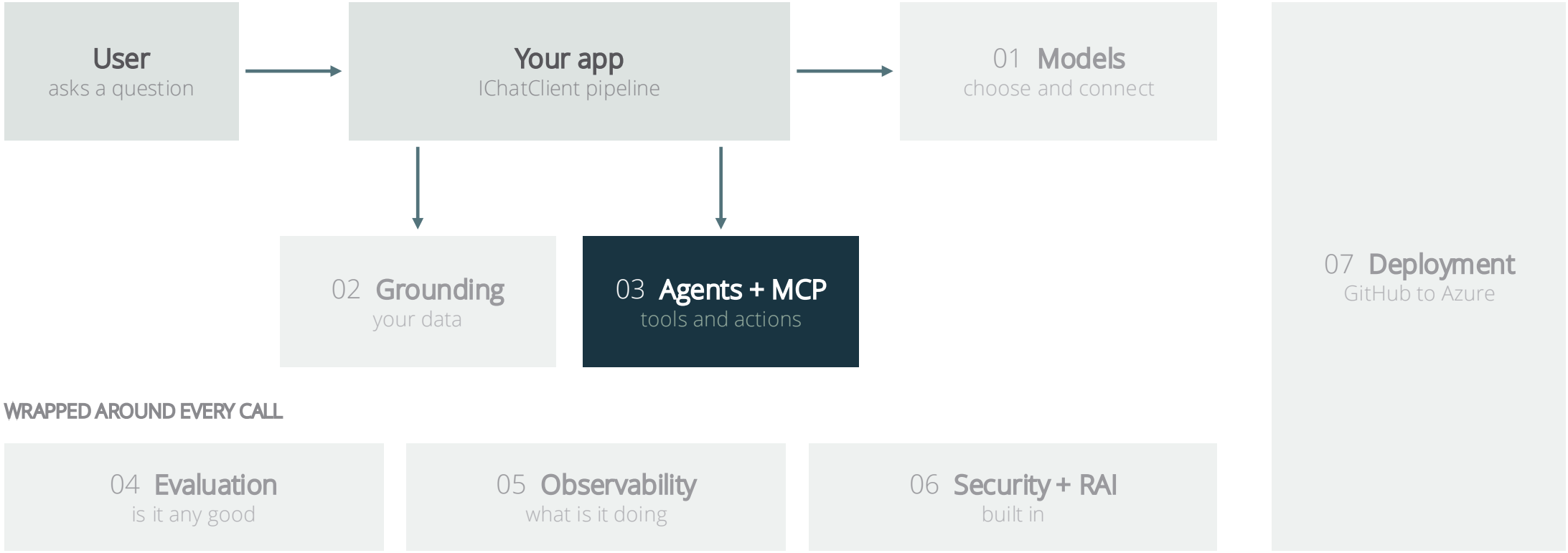


Agents and MCP

Letting the model do things

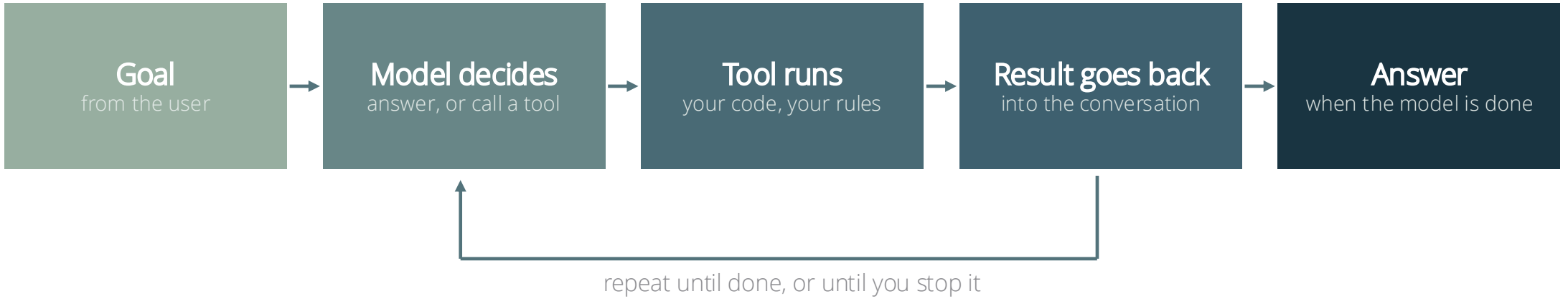
Agents and MCP

Pattern 03



What should the model be allowed to do, and how does it reach your tools?

From a reply to a loop



An agent is a model, instructions, tools, and a loop. Every pass through the loop is another model call you pay for and another decision you did not make.

MCP: one way to plug in tools

Model Context Protocol

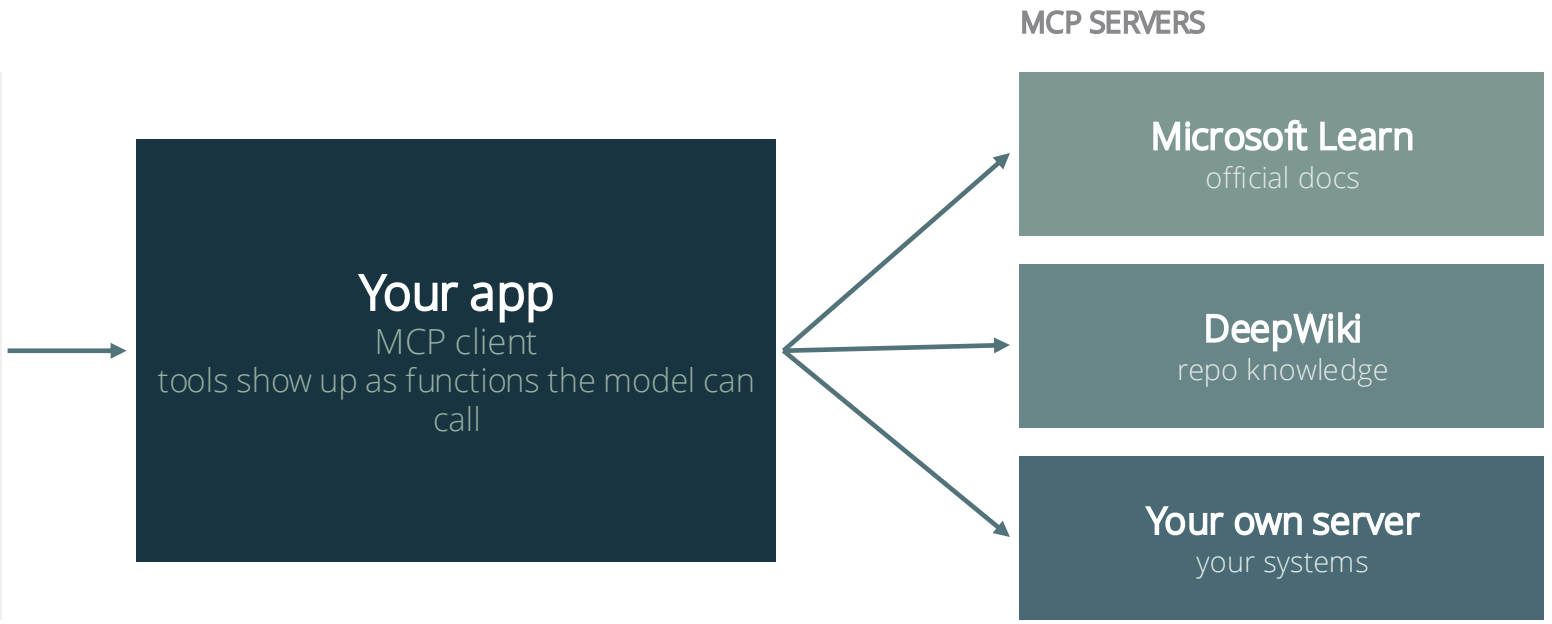
A server offers

Tools: things to do

Resources: things to read

Prompts: reusable instructions

Write a tool once. Use it from any MCP client.



A tool description is a prompt someone else wrote. Treat every MCP server as code you are choosing to trust.

Microsoft Agent Framework

The successor to Semantic Kernel and AutoGen

Agents

A model with instructions, tools, and MCP servers. Works with Foundry, Azure OpenAI, OpenAI, Anthropic, Ollama, and more.

Workflows

Functional and graph-based paths that connect agents and functions with explicit control of what runs when.

Harness agent

An opinionated agent for long, multi-step tasks: planning, context compaction, memory, tool approval.

Integrations

Model providers, tools, context providers, middleware, evaluation, and UI frameworks.

From the docs: if you can write a function to handle the task, do that instead of using an AI agent.

Agent or workflow?

Use an agent when

The task is open-ended or conversational

You want the model to plan and pick tools

One model, possibly with tools, is enough

Use a workflow when

The process has well-defined steps

You need control over the order things run

Several agents or functions must coordinate

The more you know about the steps, the less you should leave to the model. Predictable beats clever in production.

Adapted from the Agent Framework overview on Microsoft Learn.

DEMO D5

Agents and MCP

Tools, MCP, and an agent

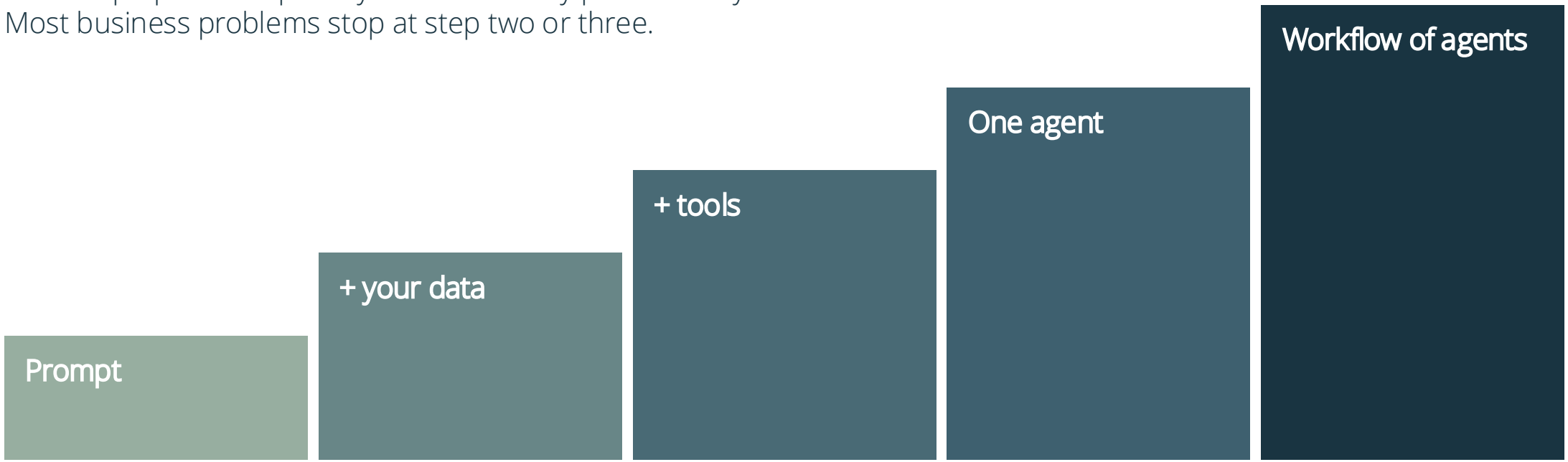
ConferencePulse · plus an Agent Framework sample

SHOW

1. A question that needs Microsoft Learn through MCP
2. Watch the tool call and the result
3. A minimal agent with one tool
4. Where the loop stops, and why

Climb only as far as you need

Each step up adds capability and takes away predictability.
Most business problems stop at step two or three.



Agents and MCP

Cost, tradeoffs, and where it fits

Lever	What you gain	What it costs you
Tool calling	The model can act on real systems	Each tool call adds a model round trip
MCP servers	Reuse tools across apps and clients	Third-party code and descriptions in your trust boundary
Agent loop	Handles tasks you did not script	Unbounded calls unless you cap them. Harder to test.
Workflows	Repeatable, inspectable multi-step runs	More design up front
Human approval on side effects	Mistakes are caught before they land	Slower, and someone has to be there

Skip until you need it: multi-agent systems, agent-to-agent protocols, and long-term memory.

Everything so far is a demo.



Evaluation

Is it any good, and how would you know?

Evaluation

Pattern 04



How do you know it is right today, and still right after you change something?

What to measure

Built-in evaluators in Microsoft.Extensions.AI.Evaluation

Retrieval

Did search bring back the right evidence?

Groundedness

Does the answer stick to that evidence?

Relevance

Does it answer the question that was asked?

Completeness

Is anything important missing?

Tool call accuracy

Did the agent call the right tool with the right arguments?

Safety

Harmful content, protected material, indirect attacks.

Quality evaluators use a model as the judge. Safety evaluators use the Foundry evaluation service. You can write your own.

Evals are tests

Run them where your other tests run



MSTest, xUnit, or NUnit. Nothing new to learn.

Built-in, plus your own with IEvaluator.

Unchanged prompt and model: served from cache.

dotnet aieval builds an HTML report.

Fail the pull request when a score drops.

Start with ten questions you know the answers to. A small set you run on every change beats a large one you run once.

DEMO D6 Evaluation

Run the evals

Chat bot · the four test cases

SHOW

1. The test project and one evaluator
2. Run it. Read the scores.
3. Break retrieval on purpose. Run again.
4. The report

Fallback Out of the oven: a finished report from an earlier run.

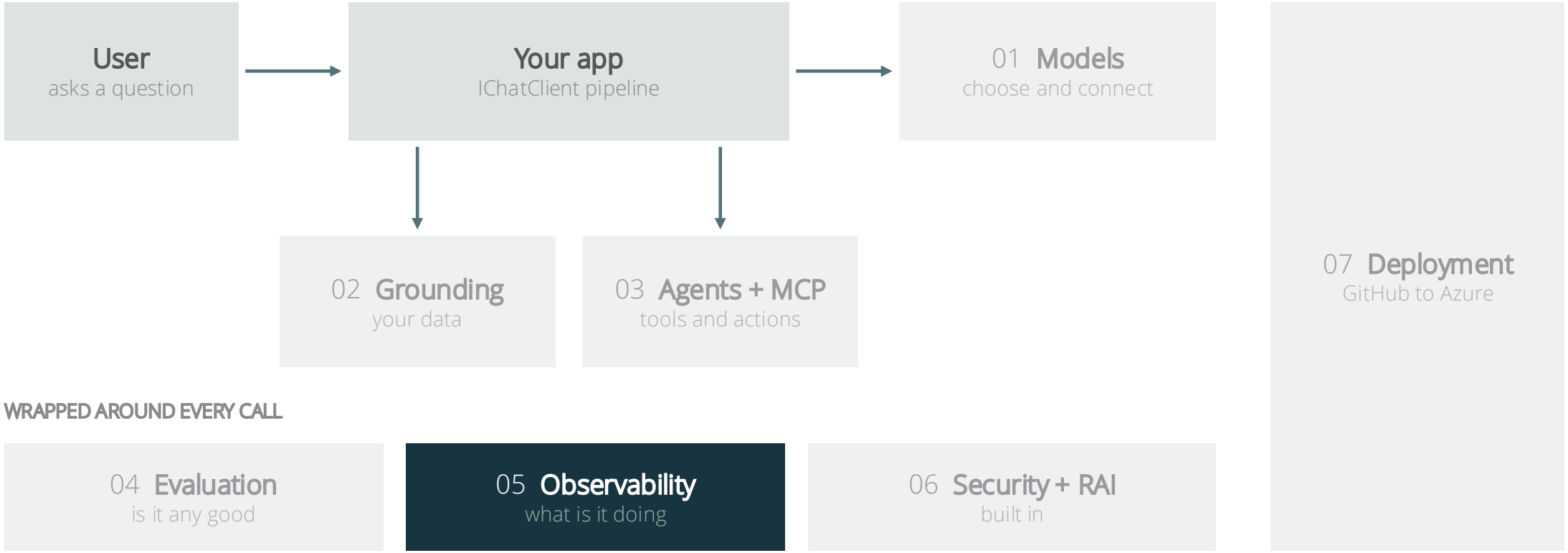


Observability

What is it doing, and what is it costing?

Observability

Pattern 05



When it is slow, wrong, or expensive in production, can you see why?

One request, end to end

What a trace of a grounded answer looks like



On every model span: model name, input tokens, output tokens, duration. Cost is tokens times price.

What to capture, and where it goes

Capture	With	Send to
Model, tokens in and out, latency Tool calls and results Retrieval: what was fetched Who asked, and for which feature	`UseOpenTelemetry()` on the chat client pipeline OpenTelemetry semantic conventions for generative AI Standard .NET logging and metrics	Aspire dashboard on your machine Azure Monitor and Application Insights in the cloud Foundry tracing for agents

Prompts and responses contain user data. Decide what you log before you turn on content capture.

DEMO D7

Observability

Follow one question through

Chat bot · Aspire dashboard

SHOW

1. Ask a question
2. Find the trace. Walk the spans.
3. Token counts on the model span
4. Per-request cost from tokens

Fallback Screenshots of a full trace with token counts.

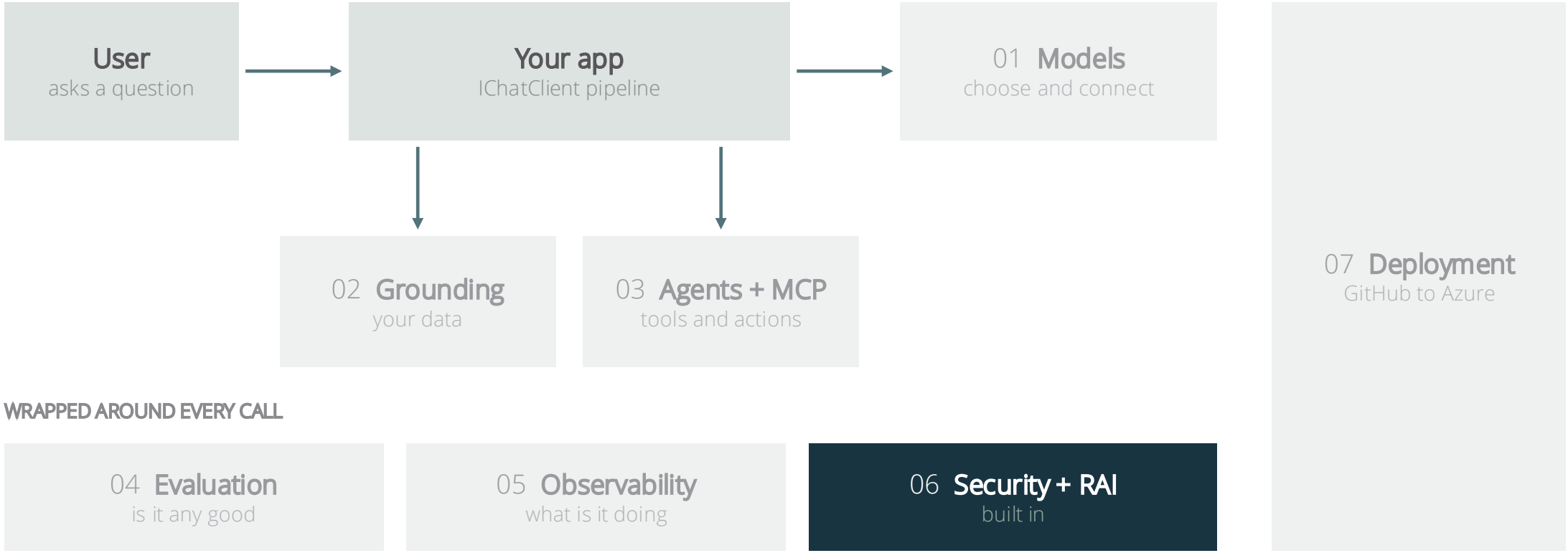


Security and responsible AI

Built in, not bolted on

Security and responsible AI

Pattern 06



What can a hostile user, document, or tool make your app do?

What is new about securing an AI app

Prompt injection

Instructions hidden in user input.

Or in a document, web page, or tool result the model reads.

Data leaking through retrieval

Search returns a chunk the caller should never see.

The model helpfully quotes it.

Too much agency

A tool that can delete, send, or spend.

Called because a paragraph of text asked nicely.

Keys and secrets

API keys in config, in repos, in prompts.

Model endpoints open to anyone with the URL.

The model cannot tell instructions from data. So your code has to: treat everything the model reads as untrusted input.

Security by design

Decisions you make once, early

1

Identity, not keys

Managed identity and Microsoft Entra ID to models, search, and storage.

2

Trim retrieval to the caller

Filter search by what this user may see, before the model sees it.

3

Least privilege for tools

Read-only by default. Approval before side effects.

4

Filter both directions

Content safety on what goes in and what comes out.

5

Secrets out of the repo

Secret scanning and push protection on. Code scanning on.

6

Log on purpose

Enough to investigate. Not a second copy of user data.

What I changed, and why

Where I started versus what is in the repos

Area	Upstream sample	My version	Why
Authentication to Azure	FILL IN	FILL IN	
Secrets and config	FILL IN	FILL IN	
Retrieval access control	FILL IN	FILL IN	
Tool and MCP permissions	FILL IN	FILL IN	
Dependencies and scanning	FILL IN	FILL IN	

Samples optimize for getting started. Production optimizes for not getting hurt. Neither is wrong.

DEMO D8

Security

Before and after

Chat bot or ConferencePulse · upstream vs. my version

SHOW

1. One change, side by side in the diff
2. Try a prompt injection against it
3. An access-trimmed query
4. Scanning results on the repo

Fallback Diff screenshots and a recorded attempt.

Responsible AI

Microsoft's six principles, and what they look like in the app

Fairness	Evaluate across the kinds of users and questions you serve.
Reliability and safety	Evals before release. Content filters at run time.
Privacy and security	Everything on the last three slides.
Inclusiveness	Accessible UI. Plain language. Works for more than one kind of user.
Transparency	Say it is AI. Show citations. Say when you do not know.
Accountability	A named owner. A human in the loop where it matters.

Principles: microsoft.com/ai/responsible-ai. The right-hand text is how I apply them.



Local or cloud?

The choice follows the way you work

Coffee break or live conversation?

Background work

Set the task and make coffee
Finish by the needed deadline

Live conversation

Brainstorm without losing flow
Pay for useful responsiveness

Shared local machine	Spread hardware use across people Allow for queues and capacity

Spend for the way you work, then measure the result

When the data must stay local

Local interface	A chat window on your machine Can still send data to the cloud
Local processing	Run the model inside the boundary Keep retrieval and embeddings there
The complete data path	Check logs and external tool calls A cloud fallback may be prohibited

If the data cannot leave, hosting must respect that boundary

Tokens: what the meter counts

A small piece of text

A word, word fragment or punctuation
1,000 tokens ≈ 750 English words

Input and output

Input: prompt, history and evidence
Output: answer and billed reasoning

The API bill

Input tokens × input price
+ output tokens × output price

Rates are usually per million tokens. A $100 chat plan has usage limits, not a fixed token allowance.

Cloud subscriptions have usage limits

A five-hour window limits how much you use, not how long you can work.

Service / plan	Short window	Other allowance	When you reach it
Claude Pro / Max	5-hour session	Weekly; model caps	Wait or paid usage credits
ChatGPT Work / Codex	Plus: 5-hour; Pro: none	Shared allowance; weekly	Check reset / eligible credits
Gemini personal app	5-hour compute budget	Weekly limit	Wait or use Flash-Lite
Provider APIs	Requests / tokens per minute	Spend and other rate caps	Retry later or raise limits

Example: first message 9am, allowance exhausted 10am, session resets 2pm.
A session reset does not clear a weekly cap. Rules checked 15 Nov 2026.

What $100 buys through the API

One call = 2,000 input + 500 billed output tokens. Daily average across 30 days.

Model	Tier	Input / 1M	Output / 1M	Calls / day	Calls / month
OpenAI GPT-6 Luna	Small	$0.1	$0.5	7,407	222,222
OpenAI GPT-6.1 Sol	Medium	$2	$10	370	11,111
OpenAI GPT-6 Astra	Large	$10	$50	74	2,222
Claude Haiku 5.5	Small	$0.1	$0.5	7,407	222,222
Claude Sonnet 5.5	Medium	$2	$10	370	11,111
Claude Fable 5.2	Large	$8	$40	92	2,777
Gemini 3.5 Flash-Lite	Small	$0.3	$2.5	1,801	54,054
Gemini 3.8 Flash*	Medium	$0.75	$3.75	987	29,629
Gemini 4 Carbon*	Large	$2	$10	370	11,111

USD, standard text rates, 15 Nov 2026. No cache, batch discount or tool fees. Budget estimates, not plan limits.
*Flash promotion ends 31 Dec; Carbon 30 Nov. Small/medium/large are illustrative service tiers.

Local hardware and simple payback

Purchase price ÷ cloud spending eliminated each month

Hardware / memory	US price	At $100 saved / mo	At $500 saved / mo
M6 Mac mini / 16GB	$899	9.0 months	1.8 months
DGX Spark / 128GB	$6,950	69.5 months	13.9 months
RTX Spark laptop / 24GB	$2,600	26.0 months	5.2 months
RTX Spark laptop / 128GB	$5,900	59.0 months	11.8 months

Before power, maintenance, setup and tax. Only valid if local models can replace that workload.
Laptop examples: Surface Laptop Ultra. Prices checked 15 Nov 2026; DGX listing out of stock.

What makes local AI pay

Cancel the $100 plan	Assume $10/month local running cost $90 saved: base mini ≈ 10 months
Keep the $100 plan	The subscription saving is $0 Local needs another reason to buy
Move some API work	Avoid $50; spend $10 locally $40 saved: base mini ≈ 22.5 months

Payback = purchase + setup cost, divided by monthly savings after local running costs.

A hosting choice you can defend

1

Data boundary

Where is this data allowed?

2

Quality

Can this model pass the tests?

3

Work pattern

Coffee break or live discussion?

4

Total cost

Hardware, service and your time

Choose the lowest-cost option that meets all four needs

Where it runs

The hosting decision

Option	You control	You pay for	Fits when
Cloud model, your tenant	Region, access, filters	Tokens or reserved capacity	Most apps. Start here.
Self-hosted model	Everything, including versions	Hardware and people	Hard sovereignty or data-boundary rules
Local on the device	The whole loop	Client hardware	Offline, private, or inner dev loop
Hybrid	Routing between them	Both, plus the routing logic	Sensitive data stays local, the rest goes to cloud

Local is a policy decision before it is a hardware decision. Ask who is allowed to see the data, then pick the box.

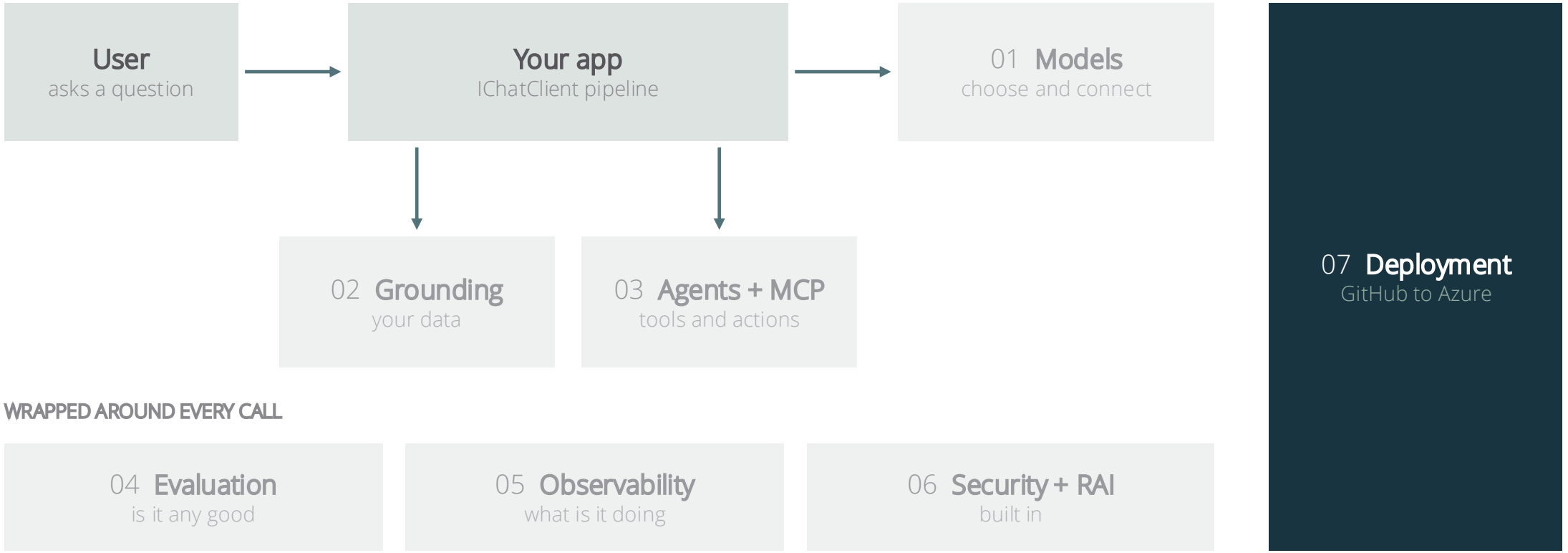


Deployment

From a repo to something people can use

Deployment

Pattern 07



Where does it run, how does it get there, and who pays for what?

From repo to running

GitHub and Azure



Repo	Pull request	Infra as code	Deploy	Run	Watch
App, infra, and evals in one place.	Build, tests, evals, code and secret scanning.	Bicep with azd. Environments you can recreate.	GitHub Actions with OIDC. No stored cloud secrets.	Azure Container Apps.	Traces, cost, and the evals again.

An AI app is still an app. The only new stage in this pipeline is the eval gate. Everything else you already know how to do.

Where the money goes

Tokens

Input, output, and reasoning.

Lever: smaller model, shorter context, cache.

Search and vectors

Index size and replicas.

Lever: chunk less, store vectors where data already lives.

Ingestion

Embedding and enrichment per chunk.

Lever: re-ingest only what changed.

Compute

The app, always on or scale to zero.

Lever: consumption plans for spiky load.

Telemetry

Traces and logs add up.

Lever: sample, and set retention.

People

Evaluating, tuning, on call.

Lever: automate the evals.

No dollar figures on purpose. Prices move. Measure your own with the trace data from pattern 05.

The tradeoff map

What to build, and what to skip

Pattern	Start with	Add when	Skip until
01 Models	One cloud model behind IChatClient	Cost or latency hurts: add a smaller model	Fine-tuning
02 Grounding	One source, simple chunks, citations	Answers miss: hybrid search, reranking	Graph RAG
03 Agents + MCP	A few read-only tools	Tasks you cannot script ahead of time	Multi-agent systems
04 Evaluation	Ten known questions, run on every change	Before each release: safety evaluators	Custom judge models
05 Observability	Traces and token counts from day one	Real users: cost per feature	Logging full content
06 Security + RAI	Identity, trimmed retrieval, filters	Tools with side effects: approvals	Nothing. Do it first.
07 Deployment	One environment, deployed from a pipeline	A team: eval gate in pull requests	Multi-region

What to take home

1	**Connect, ground, act**	Choose and connect to models, ground them in your own data, and add agents and tools, with current .NET libraries.
2	**Demo to solution**	Evaluation, observability, security by design, and deployment with GitHub and Azure.
3	**A map of tradeoffs**	Cost and hosting choices, so you can decide what to build and what to skip.

Pick one pattern to try this month. Start with ten questions you know the answers to.

Resources and credits

Workshop code

https://github.com/brianrandell/live360mco26

Slide credits

Brian A. Randell and Matt Nunn

Adapted for the web edition.

Slides licensed under CC BY 4.0.

Thank you

Microsoft, for the Azure Search OpenAI C# sample

Luis Quintanilla, for the conference assistant and the .NET Blog article

Davide Mauri, for the Azure SQL hybrid RAG sample

Matt Nunn, for the six stages and the two questions

Docs

learn.microsoft.com/dotnet/ai

learn.microsoft.com/agent-framework

So long, and thanks for all the questions

Brian A. Randell & Matt Nunn · MCW Technologies
brianr@mcwtech.com – mattnunn@mcwtech.com