Making AI Work with Your Newsroom's Data

A Practical Guide to RAG, Agents, and Fine-Tuning

Workshop · 75 min

Download PDF

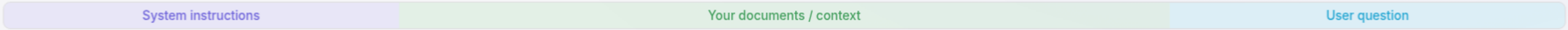
What Is an LLM?

- Trained on **massive text** — books, web, code, news articles
- Learns to predict the next token
- Builds a compressed model of language, facts, and reasoning patterns
- Only "knows" what was in its training data

GPT-4, Claude, Llama, Mistral — same core idea, different training data and techniques.

In-Context Learning

LLMs can reason over text you put **directly in the prompt** — no retraining needed.



- This is why context windows matter: 8K → 128K → 1M tokens
- Few-shot examples, documents, instructions — all work this way
- The model treats your context as if it "knew" it all along

This is the foundation — every approach we'll discuss is about getting the *right* data into that green zone.

The Gap

- Your archives, sources, internal docs — not in the training set
- LLMs hallucinate when they don't know — confidently wrong
- Recent events may be past the knowledge cutoff
- You can paste docs into the prompt, but that doesn't scale to **thousands of articles**

55,895 press releases

× avg 500 words each

=

~40M tokens

Won't fit in any context window

How do we bridge the gap?

Three Ways to Bridge the Gap

1. RAG

Search your data, inject results into the prompt. The model answers based on what you found.

Automated copy-paste

2. Agents

Give the model **tools** to search and reason on its own. It decides what to look for.

A researcher with access

3. Fine-Tuning

Retrain the model on your data so it internalizes your domain knowledge and style.

Training a new journalist

Each has different strengths, costs, and failure modes. Let's look at each one.

APPROACH 1

RAG — Retrieval-Augmented Generation

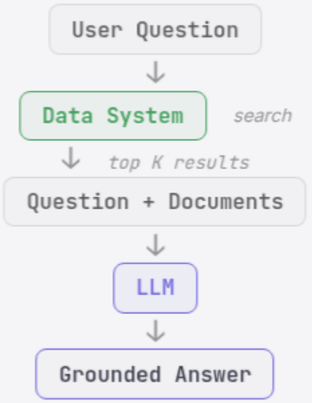
- User asks a question
- System searches your data for relevant documents
- Top results are injected into the prompt as context
- LLM generates an answer **grounded in those documents**

Think: automated copy-paste. The system finds the right documents so you don't have to.

What the LLM actually sees

APPROACH 1

RAG: How It Works



Key characteristics

- **One search**, one LLM call
- Simple, fast, predictable
- The quality of the answer depends entirely on what the search returns
- No reasoning about *what* to search for — the user query goes straight to the data system

APPROACH 1

RAG: Weaknesses

1. **Retrieval ceiling** — the answer is only as good as what the search returns
2. **One-shot retrieval** — if the first search misses, there's no second chance
3. **Query sensitivity** — slight rephrasing can return completely different results
4. **"Lost in the middle"** — LLMs pay less attention to context buried in long prompts
5. **No cross-referencing** — can't combine insights from multiple searches or follow leads
6. **Context window limits** — can only fit so many documents before quality degrades

Great for direct factual questions. Struggles with complex, multi-faceted research.

APPROACH 2

Agentic Systems

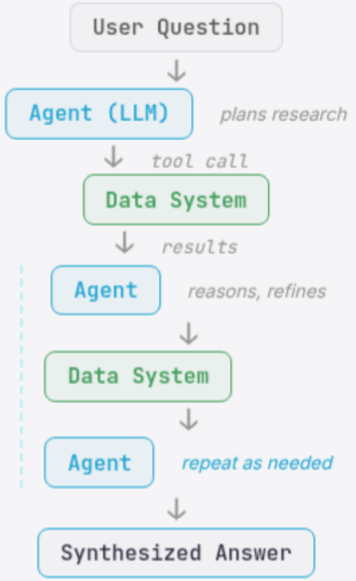
- The LLM doesn't just answer — it drives the research
- Given **tools** (search, APIs, databases) it decides what to look for
- It can refine queries, follow leads, cross-reference results
- Multiple search → reason → search cycles

Less like copy-paste, more like giving a researcher access to your archive.

```
Agent thinking:
"Let me search for covid policy in Spain..."
→ 2,438 results
"Now let me compare with Luxembourg..."
→ 1,176 results
"Interesting. Let me check the timeline..."
→ filtered by year
"Now I can synthesize an answer."
```

APPROACH 2

Agents: How They Work



Key characteristics

- **Multiple loops** — the agent controls the research process
- Can use **different tools**: search, filter, count, compare
- Each step informed by previous results
- Produces **deeper analysis** but takes longer and costs more

Tools: Claude Code, custom agents with tool use, LangChain, CrewAI

APPROACH 2

Agents: Weaknesses

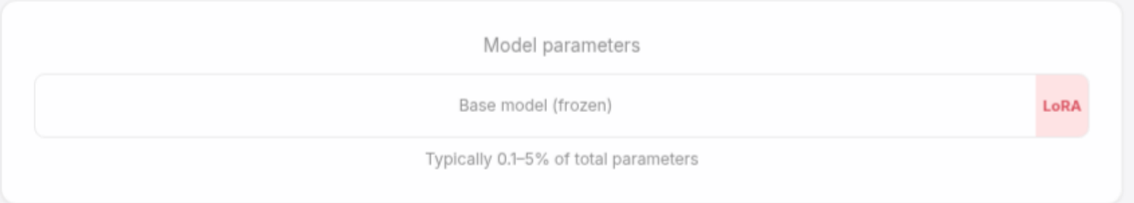
1. **Cost** — multiple LLM calls per question, 5–50× more expensive than RAG
2. **Latency** — research loops take seconds to minutes, not milliseconds
3. **Unpredictability** — same question can produce different research paths and answers
4. **Harder to evaluate** — no single retrieval step to measure; the whole chain matters
5. **Tool misuse** — the agent can search for the wrong thing or misinterpret results
6. **Compounding errors** — a wrong turn early in the chain poisons everything downstream

Powerful for complex investigations. Overkill (and expensive) for simple lookups.

APPROACH 3

Fine-Tuning with LoRA

- Instead of showing data at runtime, bake it into the model
- LoRA — Low-Rank Adaptation: train a small adapter on top of a base model
- Much cheaper than full fine-tuning — updates a **fraction of parameters**
- Shines at **simpler, focused tasks**: classification, sentiment analysis, tagging, summarization in a house style
- Can run locally — no API costs at inference time

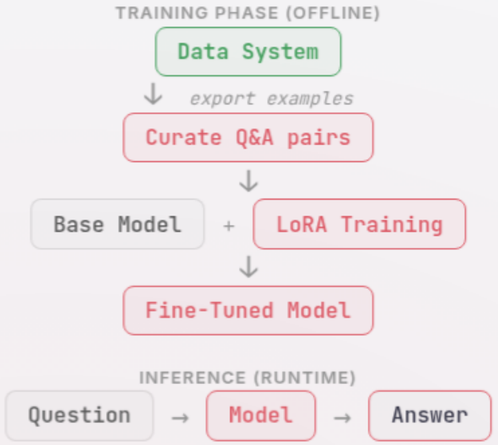


Great for newsroom tasks like:

Topic classification | Sentiment analysis | Auto-tagging | Headline generation | Language detection

APPROACH 3

Fine-Tuning: How It Works



Key characteristics

- Data system feeds **training**, not inference
- Knowledge is **in the weights** — no search at runtime
- Fast inference, no retrieval latency
- But knowledge is **frozen** until you retrain

APPROACH 3

Fine-Tuning: Weaknesses

1. **Frozen knowledge** — model only knows what it was trained on; new data requires retraining
2. **Data curation** — needs high-quality input/output pairs; garbage in, garbage out
3. **No citations** — can't point to source documents; knowledge is opaque
4. **Catastrophic forgetting** — can lose general capabilities when overtrained on narrow data
5. **Maintenance burden** — must retrain when data changes, test for regressions each time
6. **Requires ML expertise** — hyperparameters, GPU infrastructure, evaluation pipelines

Best for style/format adaptation. Not ideal when you need current facts with sources.

THE PATTERN

The Common Foundation

RAG

Query → Data System
↓
LLM → Answer

Agents

Agent ↔ Data System
↓ ↑ ↓ ↑
Agent → Answer

Fine-Tuning

Data System → Export
↓
Train → Model

Every approach starts with the same green box

Before you choose an AI approach, build a **searchable, structured, API-accessible data system**.

Get this right and you can swap AI approaches as needs evolve.

THE FOUNDATION

How the Data System Searches

Keyword Search

- Matches **words** directly in the text
- Fast, precise, predictable
- Supports filters, facets, sorting
- Struggles with synonyms and **language barriers**

```
"Klimaschutz" → 51 results (DE only)
"cambio climático" → 543 results (ES only)
"climate policy" → 0 results (no English docs)
```

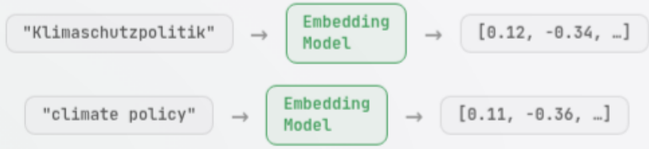
Vector Search

- Compares **meaning**, not words
- Works **across languages** automatically
- Finds semantically related content
- Requires pre-computed embeddings

```
"climate policy" → finds FR, DE, ES results
One query, all languages
```

What makes vector search work?

An embedding is a list of numbers that captures the **meaning** of a text. An embedding model converts every document into a vector.



Texts with **similar meaning** get **similar numbers** — even across languages. At search time, find the nearest vectors.

Best practice: **hybrid search** combines both — keyword precision with semantic recall.

THE FOUNDATION

Building Your Data System

The green box isn't one product — it's a role that many tools can fill. The right choice depends on what you already have.

Typesense
Fast, typo-tolerant, multilingual. Easy to self-host. Built-in vector search. Great developer experience.

Elasticsearch / OpenSearch
Industry standard. Extremely powerful and flexible. Complex to operate. Supports vectors via kNN.

Meilisearch
Developer-friendly, fast for small-to-medium datasets. Growing vector and AI search features.

PostgreSQL + pgvector
Add vector search to your existing database. Zero new infrastructure. Good enough for many use cases.

Apache Solr
Mature, battle-tested. Common in large newsrooms and archives. Dense vector support via plugins.

Pinecone / Weaviate / Qdrant
Purpose-built for vector search. Best if embeddings are your primary retrieval method. Less useful for keyword search.

All of these can be the green box. Today we're using **Typesense** — it does keyword, vector, and hybrid search in one tool.

Our Setup

Data

- Typesense — self-hosted, single container
- **55,895** EU government press releases
- 4 countries — Spain, Germany, Luxembourg, Italy
- 4 languages — ES, DE, FR, IT
- Multilingual embeddings (e5-small, 384d)
- Full-text + vector + faceted search

All data from the EU Open Data Portal. Open-licensed (CC-BY-4.0 / CC0).

Tools

- Open WebUI — RAG chat interface
- Claude Code — agentic research via API
- LLMs via OpenRouter
- Everything on **one server**

```
User → Open WebUI → Pipeline → Typesense
                        ↓
                   OpenRouter (LLM)
```

LIVE DEMO

Typesense Search

55,895 EU press releases — keyword or vector search across 4 languages

Try: covid, Klimaschuzt, climate change policy…

Keyword Vector

Type a query and press Enter

LIVE DEMO

RAG in Action

Asking questions about EU press releases — Typesense retrieves, LLM answers with citations

Ask a question → Typesense searches → LLM synthesizes → Grounded response

Open WebUI · :3000

LIVE DEMO

Agent Research

Claude Code investigates a complex question — multiple searches, cross-referencing, synthesis

`Complex question → Agent plans → Searches & reasons → Synthesized report`

`Claude Code · terminal`

Choosing Your Approach

	RAG	AGENTS	FINE-TUNING
Best for	Direct factual Q&A	Complex, multi-step research	Classification, tagging, sentiment, style
Task complexity	Simple questions	Open-ended investigation	Focused, repeatable tasks
Latency	Seconds	Minutes	Milliseconds
Cost / query	$	$$$$	Free (after training)
Data freshness	Real-time	Real-time	Stale until retrained
Can cite sources	Yes	Yes	No
Setup complexity	Low	Medium	High (ML expertise)
Needs data system	At runtime	At runtime	At training time only

These aren't mutually exclusive. A newsroom might use RAG for reporter Q&A, agents for investigations, and a fine-tuned model to auto-tag incoming articles.

Questions & Discussion

Download PDF