AI Regulation Insights Agent

Technical Design and Engineering Write-Up

1. Introduction and interpretation of the problem

When I approached this assignment, I deliberately treated it as a system design problem rather than a narrow implementation exercise. While the task describes a Generative AI "Insights Agent", the underlying challenge is about designing a system that can continuously ingest evolving information, organize it into a meaningful knowledge layer, and reason over that knowledge in a controlled and explainable way.

2. System architecture overview

The final system is organised as a modular pipeline with five clearly separated stages:

- Data ingestion and processing
- Semantic chunking
- Vector-based retrieval
- LLM generation using a multi-prompt strategy
- Interaction and memory management

Each stage is isolated behind a clear interface. This separation is intentional. In systems like this, retrieval, generation, and memory can easily blur together, making behavior difficult to reason about. I wanted each component to have a single responsibility, well-defined inputs, and predictable outputs.

This architecture mirrors the structure of production-grade insights systems: ingestion is decoupled from retrieval, retrieval is decoupled from generation, and memory is treated as an explicit design concern rather than an emergent side effect of chat history.

3. Design process and iterative refinement

The final implementation is the result of deliberate iteration rather than a single linear build.

1. I began with a straightforward baseline pipeline: scrape content, clean it, split it into fixed-size chunks, embed those chunks, retrieve relevant ones, and generate answers. While this approach worked functionally, it quickly exposed a limitation. Retrieval quality was inconsistent because important ideas were often split across chunk boundaries. For regulatory text, this led to brittle answers that were technically related but insufficiently grounded.
2. In the second iteration, I replaced fixed-size chunking with an embedding-driven semantic approach. Instead of splitting by character or token count, I tokenised the text into sentences, embedded each sentence, and grouped adjacent sentences when their semantic similarity exceeded a threshold. This change immediately improved retrieval coherence and answer quality. It reinforced an important insight: chunking strategy is not an implementation detail, but a core architectural decision.
3. In the third iteration, I focused on reasoning separation. I introduced distinct prompts for reporting, conversational question answering, and trend comparison. This removed ambiguity in prompt behaviour and made the system's outputs more predictable and explainable.
4. In the final iteration, I added persistence, memory, and interaction layers. Reports are stored as structured artefacts, recent chat turns are tracked to maintain conversational coherence, and a Streamlit interface provides visibility into system behaviour. At this point, the system felt less like a demo and more like a usable insights tool, which is why this version became the final submission.

---

4. Data collection and processing

The ingestion layer fetches articles from trusted sources, with a primary focus on GOV.UK regulatory publications. Parsing is intentionally conservative. Rather than relying on brittle selectors that often break when page templates change, I extract titles, publication dates where available, and paragraph content using stable structural markers.

Both raw HTML and cleaned text are stored. This separation supports traceability and reproducibility: raw snapshots can be reprocessed if parsing logic changes, and downstream components always operate on clean, structured text. This reflects a practical constraint often overlooked in prototypes: upstream data sources evolve, and ingestion pipelines must tolerate that change.

---

5. Semantic chunking strategy

Semantic chunking is the foundation of retrieval quality in this system.

Rather than splitting text by fixed size, I designed chunking around meaning preservation. Cleaned text is first tokenised into sentences. Each sentence is embedded using a sentence-level transformer. Adjacent sentences are grouped into a chunk when their semantic similarity exceeds a threshold, subject to a maximum chunk length. A small amount of sentence overlap is added between neighbouring chunks to preserve continuity, which is particularly important for policy documents that frequently reference the immediately preceding context.

This produces chunks aligned with conceptual boundaries in regulatory text, such as policy intent, governance mechanisms, consultation outcomes, or timelines. As a result, retrieval returns coherent context units rather than fragmented text, reducing hallucinations and improving answer grounding.

---

6. Retrieval mechanism

All semantic chunks are embedded and indexed using vector similarity search. Given a user query, the system embeds the query and retrieves the top-k most relevant chunks based on semantic similarity.

From the generation layer's perspective, retrieval returns ranked evidence. This abstraction keeps generation logic independent of the retrieval backend and allows retrieval to evolve without affecting prompt design or memory handling.

Retrieval is also designed with explicit context control. Only the most relevant chunks are passed to the language model, and conversational history is kept short. In practice, this produces more accurate answers than aggressively expanding context, which often introduces irrelevant or conflicting information.

---

7. Multi-prompt strategy and orchestration

A central requirement of the task is the use of a multi-prompt strategy, which I implemented explicitly.

The system uses three distinct prompts, each corresponding to a single reasoning mode:

- A reporting prompt that generates structured summaries from newly ingested information
- A conversational Q&A prompt that answers user questions grounded in retrieved chunks and recent context

- A trend or change-detection prompt that compares the latest report with previous reports stored in memory

Prompt orchestration follows a deterministic flow. Ingestion and chunking occur first. Reporting runs on a periodic cycle and produces persisted reports. User questions trigger retrieval and are routed to the appropriate prompt depending on intent. When a question asks about change over time, prior reports are loaded and passed to the trend prompt.

---

8. Context handling and memory design

Context selection is deliberately constrained.

For conversational queries, only the top retrieved chunks and the last few user turns are passed to the language model. This prevents token bloat and reduces the risk of irrelevant context influencing answers.

Memory is divided into short-term and long-term components. Short-term memory consists of recent chat turns and exists only to maintain conversational coherence. Long-term memory consists of persisted reports generated during periodic runs.

I intentionally avoid persisting full chat transcripts as long-term memory. Unfiltered accumulation of conversational history tends to degrade reasoning quality over time. Reports, by contrast, act as curated memory snapshots: structured, summarised, and aligned with the system's objectives.

---

9. Proof artifacts and what they demonstrate

I did not want this system to be evaluated purely on description. I designed it so behaviour is observable through concrete outputs produced during normal use.

These outputs make it possible to verify that the pipeline runs end-to-end, that answers are grounded in retrieved evidence, and that memory persists across runs.

9.1 Periodic report artefacts

Each reporting cycle generates a structured report capturing the current state of UK AI regulation and governance. Reports are timestamped and saved, forming a chronological memory.

For example, the report generated on 15 December 2025 summarises the government's ongoing stakeholder engagement following the closure of the AI regulation consultation, reiterates a proinnovation regulatory stance, and positions AI as a critical technology within the UK Science and Technology Framework.

Because reports are persisted rather than overwritten, the system can reason over change rather than restating static facts.

9.2 Retrieval-grounded conversational answers

Every user question triggers retrieval over the semantic chunk index, and responses are generated using only retrieved chunks and limited conversational context.

This behaviour is visible in the Streamlit interface, where retrieved chunks and similarity scores are surfaced. When evidence is strong, answers are precise and grounded. When evidence is insufficient, the system responds conservatively rather than speculating.

9.3 Usable long-term memory

When asking questions such as "How has the picture changed since last week?", the system loads recent and prior reports and generates a comparison describing shifts in emphasis, tone, or stage of policy development.

This demonstrates that memory is implemented through persisted artefacts reused for reasoning, not simulated through conversational history alone.

---

10. Deployment approach on AWS

The local implementation maps cleanly onto a practical AWS deployment without changes to core logic.

In AWS, I would treat the system as a scheduled ingestion and reporting pipeline plus a continuously running query service. Data and reports would be stored in S3. EventBridge would trigger hourly reporting runs, with Step Functions orchestrating ingestion, chunking, embedding, and report generation.

Embeddings would be generated via SageMaker, and retrieval would use either a managed vector store or a versioned in-memory index depending on scale. The chat interface would run as a containerised service on ECS, reading reports from S3 and querying the retrieval backend. Monitoring and logging would be handled via CloudWatch.

This mirrors the same architecture used locally, replacing local components with managed services while preserving clear boundaries.

---

11. Conclusion

I approached this assignment with the same mindset I apply to real-world AI engineering problems: clarify intent, explore alternatives, converge deliberately, and prioritise clarity and correctness over novelty.

The final system ingests real regulatory information, chunks it semantically, builds a vector-based retrieval layer, generates structured reports, supports grounded conversational Q&A, and maintains memory over time through persisted reports. More importantly, it does so in a way that is explainable, testable, and extensible.