Argusa

Engineering reliable RAG systems

Lessons from a Hackathon

Table of content

Context and scope

Organizations are rapidly exploring Retrieval-Augmented Generation (RAG) to extract value from their internal knowledge and accelerate decision-making. Yet most early initiatives stall at the proof-of-concept stage. They underestimate the engineering depth required to build a RAG system that is reliable and secure, while keeping aligned with business constraints. This white paper presents insights gained from an AI hackathon, which was first developed and tested internally, before being exposed to a broad audience of dozens of participants during the Argusa AI Challenge. The objective of developing a full multi-phase process was to stress-test real-world RAG challenges and consolidate methodological foundations for enterprise deployment.

The approach began by generating a realistic synthetic document corpus that mimics the complexity of actual enterprise data. This allowed experimentation without exposing sensitive information. The first phase of internal testing, before the competition, let us implement complete RAG systems across multiple ecosystems, offering a broad spectrum of techniques to compare. Finally, we designed a full evaluation framework that mixes extended automation and human proofreading as guardrail.

For enterprises, the value is clear: a well-engineered RAG accelerates access to knowledge, enhances decision workflows and supports governance by grounding outputs in verifiable sources. This hackathon consolidated our theoretical knowledge and capability to design production-ready RAG systems, navigate architectural choices, and anticipate operational risks. This document provides a pragmatic blueprint for organizations seeking to move from experimentation to deployment.

The hidden complexity of enterprise RAG

The promise and the reality

Retrieval-Augmented Generation promises grounded answers by connecting LLMs directly to internal documentation. The value proposition is compelling: leverage existing knowledge and deploy intelligent assistants without retraining models. Yet most enterprise RAG initiatives stall at proof-of-concept. Real environments contain noisy, contradictory, and incomplete information spread across heterogeneous systems. Access is governed by strict security and compliance rules. Under these conditions, RAG becomes an engineered chain of interdependent components where weakness in any stage degrades the entire system.

A structured exploration across multiple architectures

We designed a hackathon to expose some of these difficulties systematically. We built end-to-end RAG systems on a synthetic but realistic corpus mimicking a manufacturing company's documentation landscape. The solutions proposed during the phase of internal testing used different stacks. The goal was not to identify a single "best" technology but to explore patterns, failure modes, and robust practices that generalize across different environments.

From experimentation to production readiness

The exercise surfaced a recurrent blind spot: evaluation of RAG systems is intrinsically difficult. Traditional metrics don't capture answer quality, LLM-based evaluators introduce variability, and human review doesn't scale. The hackathon therefore devoted significant effort to building an automated evaluation pipeline with explicit rubrics and safeguards. This document translates these experiments into a pragmatic guide for moving from promising POCs to production-ready systems that operate under enterprise constraints.

Generation of the corpus

Building a RAG system on enterprise data for a hackathon purpose requires full access to internal documentation. Yet such access is rarely permissible. Even a small subset of real corporate files contains confidential information, strategic content, or personal data that cannot be exposed in a shared experimental environment. For the hackathon, this constraint was absolute. No team could work with real documents. Even anonymisation would have been insufficient, as structural, contextual and behavioural metadata can still leak sensitive information. The only viable option was to generate a synthetic corpus that preserved the complexity of real data while ensuring complete security.

Maintaining coherence across document generation

Designing the corpus required establishing a coherent fictional company: products, staff, projects, and processes. Initial iterations with an LLM (ChatGPT) defined the core narrative and organizational structure. However, insufficient upfront structure created contradictions when generating additional documents and propagated inconsistencies throughout the corpus. Documents were therefore generated one by one, with previously generated material re-injected into context to maintain consistency across emails, reports, and project updates. This iterative

process revealed what also occurs in real enterprises: weak documentation governance means RAG systems inherit underlying ambiguity.

Manual oversight remained necessary to correct deviations and ensure overarching coherence, reinforcing a structural insight: enterprises cannot rely on unsupervised large-scale ingestion. Human validation remains essential, even in AI-augmented workflows.

Deliberately introducing realistic complexity

A second design principle was deliberate inclusion of difficult and noisy data.

- Real-world corpora contain ZIP files, emails with complex headers, inconsistent dates, partial information, redundancies, and generic titles like "Introduction." The synthetic corpus reproduced these signal-to-noise conditions to force teams to handle realistic ingestion challenges.
- Images were included in the corpus, which forces the implementation of an image recognition tool. This task is technically complex, but essential for a proper interpretation of diagrams, charts and tables.
- Some documents required cross-referencing across multiple files—a pattern that often exposes weaknesses in retrieval strategies.

A foundation for reliable experimentation

The final corpus contained approximately one hundred files across PDF, DOCX, PPTX, TXT, YAML, JSON, logs, emails, and archives. Information was heterogeneous, partially redundant, and sometimes intentionally contradictory, as production RAG systems must operate under these conditions.

Developing a RAG Solution

Exploring RAG patterns across diverse toolchains

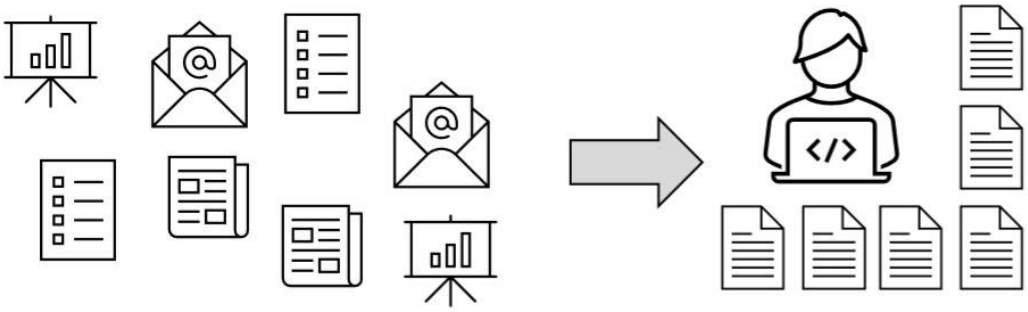
Five consultants developed independent RAG solutions during the challenge design phase, each exploring different technical stacks to validate feasibility and iteratively refine the competition structure. The toolchains included: Snowflake Cortex, Databricks, Weaviate, ChromaDB, and custom Python pipelines, e.g. with LangChain orchestration. This diversity exposed different behaviors across ecosystems, clarifying the practical impact of architectural choices. No single pipeline structure proved universally superior: performance depends on data characteristics, infrastructure constraints, and business requirements.

Answer accuracy ranged from 65% to 80% across solutions, reflecting natural variability in retrieval strategies, chunking quality, and model selection. The exercise strengthened our capacity to design client-grade systems while understanding their inherent trade-offs and interdependencies, thus establishing a robust foundation for designing client-grade RAG systems.

The RAG pipeline architecture the emerged out of these experiments is globally technology-agnostic: overall, solutions converge on the same six foundational steps. Tools differ in implementation detail, but the underlying tradeoffs. The following sections outline these steps and the compounding design decisions embedded within each.

Technical pipeline of a solution

Step 1 : Ingestion

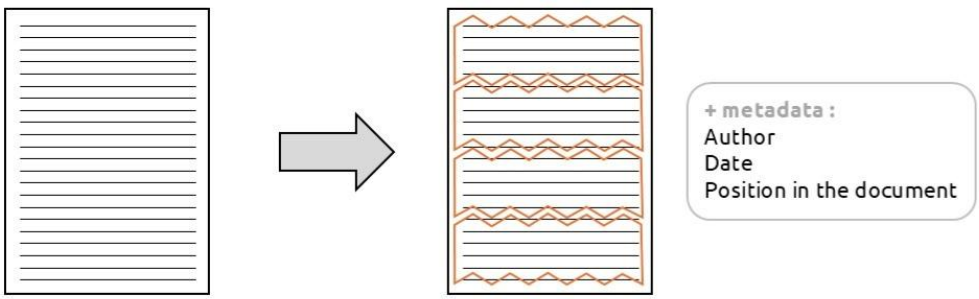


During the ingestion stage, the pipeline converts a diverse set of raw files into exploitable textual documents. This phase focuses on capture rather than transformation: every item in the corpus (slides, emails, reports, lists, semi-structured files) is detected, loaded, and expressed as complete text, without yet altering or cleaning the content. The goal is straightforward: ensure that everything fed into the system exists in a readable, manipulable form, regardless of its original format.

A file that is not or poorly ingested is knowledge the system will never possess, regardless of how sophisticated retrieval or generation become. This makes ingestion the highest-leverage failure point in the pipeline. A 95% ingestion rate might seem acceptable, but if those missing 5% contain critical information, downstream performance becomes irrelevant.

Two key challenges appear at this stage. The first is full coverage of file types. Some formats ingest cleanly, while others require specific logic (archives, multipart emails, noisy PDFs). If a file is not ingested, the RAG loses that information permanently. The second challenge is traceability. The system must build a complete inventory including paths and minimal metadata. This visibility is essential to audit the pipeline's behavior, diagnose gaps in retrieval, and maintain coherence across the entire chain.

Step 2 : Pre-processing



During the pre-processing stage, the textual contents produced by ingestion are standardized and segmented into coherent text chunks. The schema illustrates this transformation: a full document is first normalized, then divided into multiple segments, each enriched with metadata such as

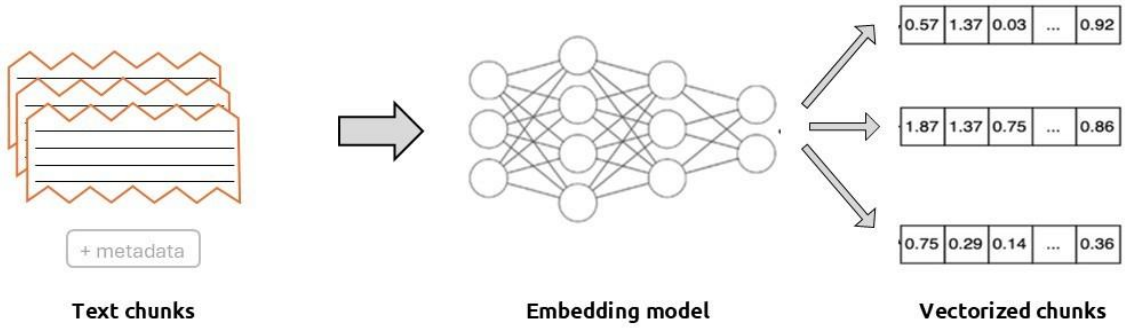
author, date, and position within the document. This step creates structural units and ensures that the system works with consistently formatted pieces of information rather than large, irregular files that exceed model limits or contain internal noise.

This stage introduces two major challenges. The first challenge concerns **metadata quality**. Metadata is not decorative: it guides filtering, improves interpretability, and supports citation or auditing. If metadata extraction is incomplete, noisy, or inconsistent, downstream stages lose crucial context about provenance, chronology, or document scope. The second is **avoiding over-fragmentation**. Splitting too aggressively produces many small, low-value chunks that dilute semantic meaning, increase storage volume, and degrade retrieval quality. The system must therefore balance chunk size, textual coherence, and the natural structure of the underlying document to maximize the signal-to-noise ratio.

The pre-processing step exposes a fundamental trade-off: chunks must be small enough for precise retrieval but large enough to preserve meaning. More than a hyperparameter problem, it is an inherent tension where optimizing one dimension degrades the other, forcing explicit tradeoffs between retrieval precision and semantic coherence.

Pre-processing is therefore the moment where raw text becomes structured knowledge, ready for semantic indexing.

Step 3 : Indexation



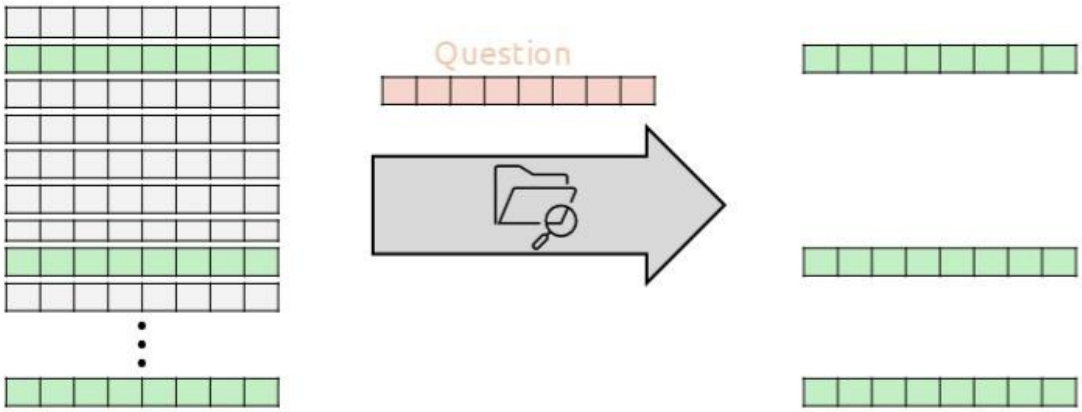
During the indexation stage, each text chunk produced during pre-processing is converted into a vector representation using a semantic embedding model. The schema shows this flow clearly: chunks enriched with metadata enter the embedding model, which outputs numerical vectors capturing their semantic meaning. These vectors populate the index that will power retrieval. The size depends on the model, but for typical OpenAI embedding models, the vector size spans from 1024 to 3072.

One challenge is the choice of embedding model. There is no golden rule and the choice will depend on the use case. However, the model used at indexing time must be consistent with the one used to embed queries, otherwise, semantic distances become unreliable and retrieval degrades. The second challenge concerns volumetry: if chunks are too small, the index grows rapidly, increases storage costs, and amplifies noise. If chunks are too large, key information becomes harder to retrieve. The design must find a balance that preserves meaning, avoids redundancy, and remains computationally reasonable.

Indexation crystallizes three compounding tradeoffs. First, embedding model selection locks in retrieval behavior. Second, vector dimensionality forces a choice between semantic fidelity and operational cost: higher dimensions capture nuance but scale poorly. Third, chunk size interacts with both: smaller chunks demand more vectors, amplifying storage and latency issues, while larger chunks dilute the precision gains from high-dimensional embeddings. These strategies are tightly dependent and optimizing for one objective typically sacrifices another.

This step is therefore the bridge between raw text and efficient search, enabling the system to compare a user query with the most relevant parts of the corpus. This is the crucial step where the RAG system builds its semantic memory. The quality of this memory determines the quality of all future answers.

Step 4 : Retrieval



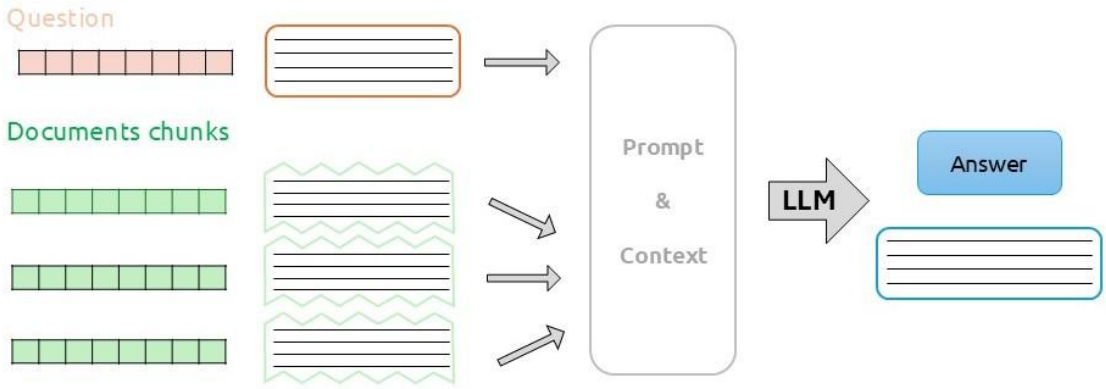
During the retrieval stage the system receives a user question, embeds it, and compares this vector to the indexed vectors to select the most relevant text chunks. The schema visualises this process: the question vector enters a search mechanism that ranks and extracts the top chunks expected to help answer the query. Retrieval only relates to selecting context and does not generate text. Its purpose is to gather the right pieces of information before any reasoning occurs.

Many technical degrees of freedom make this step as complex as crucial. First, hybrid retrieval (i.e. balancing semantic similarity and keyword matching) is often required to capture both conceptual meaning and literal details such as names, dates, or product codes, but the balance between both aspects can be shifted. Second, the number of retrieved chunks matters: too few reduces recall, too many introduces noise and inflates LLM costs. Third, the informativeness of the chunks is critical. Even if semantically close, a chunk may be unhelpful if it contains generic or incomplete content.

Retrieval is where upstream decisions converge and failures compound. This stage acts as a quality revelator: poor chunking surfaces as semantic orphans, weak metadata eliminates filtering options, and embedding mismatches distort ranking. Whatever context fails to surface here is permanently lost to the generation stage.

Retrieval is the system's decision-making layer: it determines what the model will "see" before generating an answer.

Step 5 : Generation



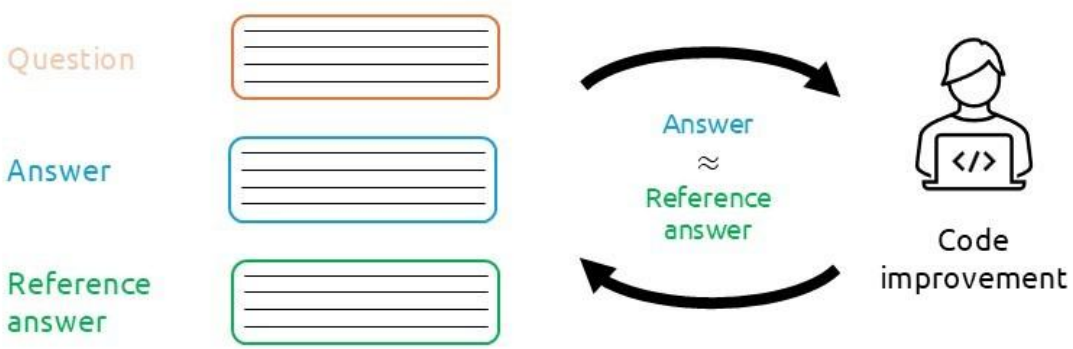
During the answer generation stage, the system constructs a prompt that combines the user question with the retrieved text chunks. This combined context is then passed to the LLM, which produces the final response. The schema above highlights this flow: the question and document segments converge into a prompt block, which the model transforms into an articulated answer. This step is where reasoning, synthesis, and formatting occur.

The LLM must synthesize the retrieved information into a clear and structured answer while ideally maintaining traceability (refer to sources). Even more importantly, the system must avoid hallucinations by constraining the model to rely exclusively on the provided context, an essential requirement for enterprise reliability. Finally, it must handle uncertainty: if information is missing, ambiguous, or contradictory, the model should avoid fabricating details and instead signal insufficient evidence.

Generation involves a critical tradeoff between confidence and utility. Forcing uncertainty signals and citations reduces hallucination but increases hedging, answers become cautious and less actionable. Conversely, confident synthesis improves user experience but risks plausible fabrications and thus wrong answer with respect to the enterprise corpus. Whereas stricter guardrails produce safer outputs, looser constraints improve fluency. This tradeoff cannot be eliminated and the balance depends on use case tolerance for error versus need for decisiveness.

Prompt design is therefore a core engineering task, shaping the model's behavior and ensuring alignment with business expectations. Answer generation is where the RAG demonstrates its added value, through grounded and contextual outputs.

Step 6 : Auto-evaluation



During the internal evaluation stage, the system compares the generated answer with a reference answer, iterating until the performance becomes satisfactory. The reference answer can either come from the hackathon organizers (in our case) or more broadly from business experts in a company. This mirrors a real deployment situation, where evaluation is essential to detect weaknesses before exposing the system to users.

This last step aims to detect quality defects early. Issues may stem from ingestion, chunking, retrieval or prompting. Evaluation must reveal them before the “competition”, before the system goes live. The implementation of the evaluation must separate retrieval performance from generation performance: a wrong answer may come from missing context or from incorrect reasoning and the evaluation pipeline must distinguish the two. The split between retrieval and generation errors is not trivial. Two possible directions, to be adapted to specific constraints and use cases, are:

- Add a verification step after generation to validate that the produced response is grounded in the provided context. This primarily highlights generation errors.
- Introduce an intermediate development step where, instead of generating an answer, the system is required to produce strict source citations only. This assesses retrieval quality while excluding generation quality.

The other quality defect that can be prevented during this step is robustness issues: answers should remain stable when input wording varies or when documents contain partial inconsistencies.

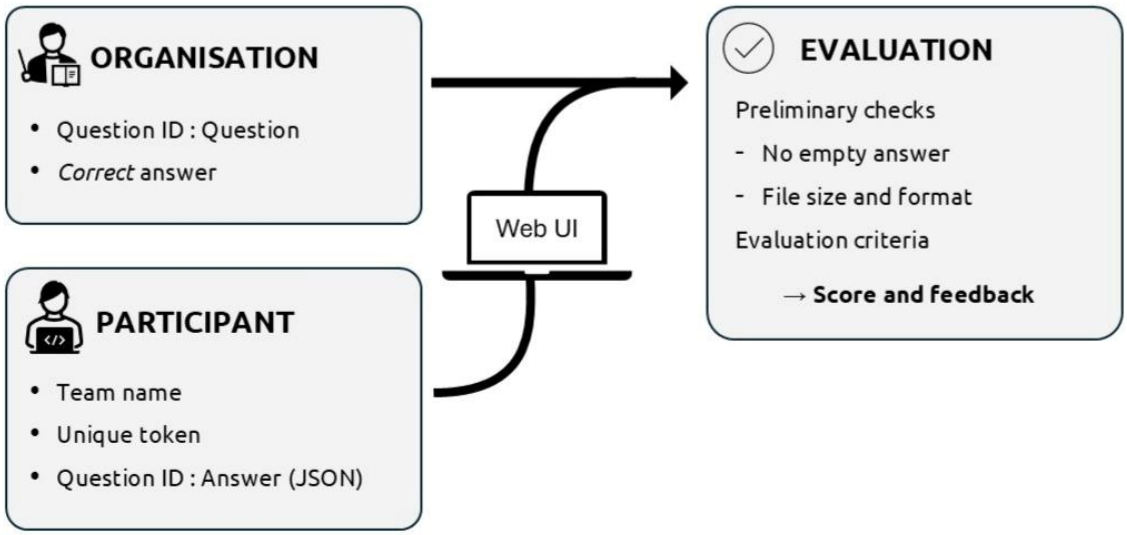
Without auto-evaluation, the developer iterates blindly, modifying the different steps without knowing if quality improved or regressed. The tradeoff is evaluation rigor versus iteration speed: comprehensive testing provides confidence but slows development. Effective evaluation must also isolate which stage failed, to guide targeted fixes. This feedback loop transforms ad-hoc experimentation into disciplined engineering.

Internal evaluation closes the loop: it validates the pipeline, guides improvements, and ensures trust in the final RAG system.

Designing a scalable evaluation pipeline

Evaluating RAG systems at scale requires an end-to-end pipeline that manages submissions, guarantees fairness, and controls the variability of LLM-based judgements. For the hackathon, we implemented such a pipeline, from participant interface to final scoring. This section describes how it works and why it matters for enterprise RAG deployments.

Participant interface and secure submission



During the actual hackathon, the participating teams interacted with the evaluation system through a dedicated web interface. Each team received a unique token linked to its identity. Submissions were accepted only once per token, preventing duplicate or modified entries after deadline. The UI guided participants to upload a JSON file containing a mapping from question IDs to answers, enforcing basic format, size, and completeness checks before acceptance.

In addition, multiple guardrails are implemented: tokens guarantee single-use submissions, and all interactions are logged with question IDs and team identifiers, and finally answers are sanitised before evaluation for instance by preventing from prompt injection attack,

This approach ensured a simple user experience while protecting the backend from malformed payloads and unintended re-submissions. On the organiser's side, the system maintained a catalogue of questions, their identifiers, and reference answers, forming the backbone of the evaluation dataset.

The evaluation pipeline

Reference Framework and Evaluation Criteria

For each question, the organisation defined a so-called *reference answer* that captures the expected factual content and level of detail. However, this principle is in itself debatable. A single “exact” answer inevitably simplifies reality and represents a known risk of the model: some valid formulations may differ from the reference while still being acceptable.

The evaluation criteria are structured along three dimensions, each with its own weighting:

- **Accuracy (50%)** : No incorrect or hallucinated information.
- **Completeness (30%) :** Coverage of all key elements required to answer the question.
- **Conciseness (20%) :** Clarity and brevity, penalizing unnecessary verbosity.

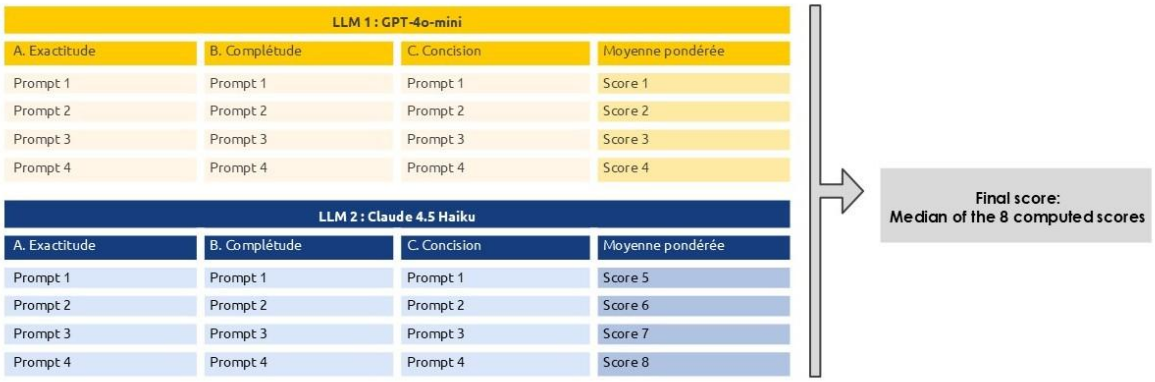
Each criterion was scored on a 0–5 scale. The LLM was instructed not only to assign scores but also to generate textual feedback explaining its judgement, highlighting omissions, inaccuracies, and stylistic issues.

LLM-Based evaluation with prompt variants

Before evaluation begins, submitted answers undergo pre-cleaning to remove prompt-injection patterns, and single-use tokens prevent duplicate submissions. These guardrails protect the scoring logic from manipulation and ensure evaluation integrity. The evaluation architecture then operates along two dimensions to mitigate LLM variability: self-consistency within a single model, and cross-validation between independent models.

- **Self-Consistency** Four different prompt variants are evaluated by the same LLM, each producing independent weighted scores. This approach reduces the impact of poorly formulated prompts or single-run instability, ensuring that scoring reflects the answer's quality rather than prompt sensitivity.
- **Dual-Model Aggregation** The same four prompts are sent to a second, independent LLM, yielding four additional scores. This cross-model validation guards against model-specific biases or idiosyncrasies, ensuring robustness across different reasoning patterns.

Each answer thus receives eight weighted scores, four from each model across four prompt variants. The final evaluation score is the median of these eight values, minimizing the influence of outliers while preserving central tendency.



For reporting, this median is paired with representative feedback from the run whose score is closest to the final value, providing teams with both quantitative benchmarks and qualitative explanations.

Where RAG breaks: A dive into the most challenging questions

Several questions from the challenge systematically revealed structural weaknesses in the RAG pipelines. We selected three examples to illustrate it.

1. **Exhaustive list questions:** When a question required *"list all…"* elements matching a criterion, retrieval often captured only a subset. Fixed-k similarity search missed lower-salience items, leading to incomplete answers, even when the information existed in the corpus.
2. **Hidden or implicit information:** Questions about '*Python libraries'* showed that answers sometimes depended on concepts never explicitly named in the documents. Libraries like NumPy or Sklearn were mentioned, but not labelled as "Python libraries", causing retrieval to fail despite relevant content.
3. **Reasoning traps:** Regarding a company's presentation, the question *"When was Camille invited to present?"* exposed reasoning fragility. Camille never presented; she only communicated about the event. Systems unable to detect this nuance produced confident but incorrect answers.

These cases highlighted core risks: retrieval fragility, semantic mismatches, implicit signals overlooked by embeddings, and the persistent difficulty of avoiding hallucination when documents contradict naive expectations.

Human-in-the-loop review

A final control concerns discrepancies: if the spread between the two LLMs' scores is not "reasonable" or if internal variance between prompt runs is too high, the system raises an alert. These flagged cases are escalated to manual review, where a human judges the answer using the LLMs' textual feedback as a guide.

This human-in-the-loop mechanism ensures that unstable evaluations, edge cases, or ambiguous questions do not distort the ranking. It also mirrors what enterprises will need in production: automated evaluation for scale, complemented by targeted human oversight where confidence is low or stakes are high.

RAG evaluation faces an inherent paradox: the systems used to evaluate RAG suffer from the same limitations as the systems being evaluated (variability, hallucination risk, and sensitivity to prompt phrasing). This validation gap makes human oversight become non-negotiable, not as a fallback but as a structural requirement to interpret edge cases, calibrate scoring thresholds, and catch systematic blind spots that automated metrics miss. Evaluation thus mirrors the broader RAG challenge: automation enables scale, but reliability requires human judgment in the loop.

Scaling evaluation without ground truth

RAG systems generate open-ended answers where multiple valid responses exist, eliminating the possibility of binary correctness checks. Traditional evaluation metrics assume a single ground truth, but RAG answers vary legitimately in completeness, phrasing, and emphasis. This creates a scalability challenge: human review provides

nuanced judgment but cannot scale across thousands of queries, while automated LLM-based scoring scales efficiently but introduces variability and potential unreliability.

When answers admit multiple valid formulations or have poorly defined boundaries, only human review can calibrate scoring thresholds and catch systematic blind spots. In this context, periodic evaluation campaigns become essential: regenerating reference answers from updated models or business experts establishes evolving benchmarks that reflect current expectations rather than static ground truth. For stricter domains, additional mechanisms, such as user feedback loops for domain, extend this framework. All share the same principle: automation enables scale, human judgment maintains trust.

From learnings to roadmap

RAG as a coherent engineering system

The hackathon taught us that a RAG pipeline is only as strong as its weakest link. Each stage contributes to the final output in ways that cannot be isolated. A coherent corpus remains essential: incomplete metadata, inconsistent structures, or noisy documents propagate downstream into unstable retrieval and ambiguous answers. The same applies to preprocessing choices, where chunk size, segmentation strategy, and metadata extraction determine the quality of the indexed representation.

The engineering reality of RAG is a sequence of trade-offs :

- Larger chunks improve context but blur precision; smaller ones boost recall but increase noise and cost.
- Retrieval strategies must balance semantic similarity and literal matching.
- Model selection impacts both price and accuracy.

Scaling such a system in production obviously requires robust performance tuning. But it also implies real-time corpus synchronization, distributed architectures for large volumes, and standardized metadata to maintain consistency across updates. The pipeline must evolve as a single system, not a collection of components.

Retrieval and intelligence: the real performance lever

Most of the hackathon's difficult questions revealed the same insight: retrieval governs performance. When questions required exhaustive lists, retrieval often surfaced only the most salient items, leaving long-tail information undiscovered. When answers relied on implicit cues—such as identifying "Python libraries" not explicitly labelled as such—pure semantic search underperformed. Reasoning traps, like attributing a presentation to someone who only announced it, showed that incomplete retrieval encourages the model to fill gaps with plausible but incorrect inferences.

These limitations outline the roadmap for more intelligent systems. Hybrid retrieval with re-ranking reduces blind spots; metadata-aware search restores precision on literal details;

automatic extraction of filters by LLMs allows the system to refine queries dynamically. Multi-hop retrieval and agentic behaviors extend this further by enabling the system to navigate documents, aggregate clues, or request clarifications. Moving beyond static similarity search is essential: the next generation of RAG will be defined by retrieval strategies that reason, not merely retrieve.

Evaluation and governance as foundations for trust

Evaluating RAG output remains one of the field's hardest problems. No single metric captures quality, and evaluator LLMs introduce variability. The dual-model, multi-prompt evaluation pipeline built for the hackathon demonstrated how structured scoring, consistency checks, and human review together improve reliability. Continuous evaluation is not optional, it acts as the foundation for identifying regressions, detecting weak retrieval patterns, and ensuring that improvements at one stage do not degrade another.

Equally important is governance. Enterprises require audit logs, permission management, provenance tracking, and sensitivity controls to ensure safe deployment. Quality dashboards and user feedback loops help maintain trust and operational visibility. As RAG systems become embedded in business processes, evaluation and governance will determine both technical performance and organizational acceptance.

Conclusion

This hackathon demonstrated that deploying a RAG system requires expertise across three interconnected domains.

First, the ability to generate realistic synthetic corpora. While not part of the RAG pipeline itself, this capability is essential for a hackathon context as well as for safe experimentation in a corporate environment: it reproduces enterprise data complexity while ensuring confidentiality, enabling teams to stress-test systems before production deployment. Second, engineering robust RAG pipelines where ingestion, retrieval, and generation function as an interdependent chain. Weakness at any stage degrades the entire system. Third, rigorous evaluation through structured, variance-controlled methods that produce trustworthy performance metrics.

Together, these capabilities allow us to move RAG from promising prototype to dependable production system. The path requires understanding where retrieval fails, how context shapes generation, and which architectural choices matter under real constraints. This hackathon systematically exposed these failure modes and built the foundation for designing RAG solutions that are robust, auditable, and aligned with how businesses actually operate.