CHECKLIST

Scoping a RAG or AI assistant project: a checklist

30 questions to settle before building a RAG system or AI assistant: use case, data, access control, retrieval, model choice, evaluation, operations and go/no-go.

30 checks Tick each item once it is in place. The online version has the same explanations, with links.

Online version: sigmacode.io/en/insights/rag-ai-assistant-scoping-checklist

Most AI assistant projects that disappoint do not fail because of the model. They fail because nobody agreed on what the assistant is for, the underlying content was incomplete or outdated, permissions were an afterthought, or there was no way to tell whether a change made things better or worse. Model choice matters, but it is usually one of the easier decisions.

This checklist collects the questions we work through before building a retrieval-augmented generation (RAG) system or an AI assistant on company data. Use it to prepare an internal project, to brief a vendor, or to check whether a proposal you received covers the essentials. You do not need every answer on day one, but you should know which ones are still open.

1. Use case and success criteria

Write these down before anyone opens a notebook or compares models.

- ☐ 01 **One primary job.** Name the single task the assistant must do well first, for example answering support agents' product questions or finding clauses in supplier contracts. Further use cases can follow once the first one is measured; mixing several at the start blurs both requirements and evaluation.
- ☐ 02 **Users and their situation.** Describe who asks, how often, in which language and on which device, and what they do with the answer. An internal expert who checks sources needs something different from a customer who acts on the answer directly.
- ☐ 03 **Real questions with ideal answers.** Collect 20–50 real questions from tickets, emails or chat logs, and have a domain expert write the ideal answer and name the source it should come from. This set defines what "good" means and becomes the seed of your evaluation set.
- ☐ 04 **Out of scope and refusals.** List what the assistant must not do: give legal, medical or financial advice, answer beyond its sources, make commitments on behalf of the company. Decide what it should say instead and where it should point the user.
- ☐ 05 **Baseline of the current process.** Record how the job is done today, how long it takes, who does it and what it costs. Without a baseline, "the assistant helps" is an opinion rather than a result.

2. Data sources

The quality of the answers is capped by the quality and accessibility of the content behind them.

- [ ] 06 **Source inventory.** List every source the assistant should use: wikis, SharePoint or Google Drive folders, ticket systems, databases, PDFs, websites. For each one, note how it can be accessed (API, export, crawl) and whether that access is permitted.
- [ ] 07 **Formats and document quality.** Check for scanned PDFs that need OCR, complex tables, forms, slides and images that carry essential content. Tables and scans are where naive pipelines lose the most information, so sample them early.
- [ ] 08 **Ownership and freshness.** Name an owner for each source and record how often it changes. If nobody owns the content, nobody will fix the wrong answers that come from it.
- [ ] 09 **Duplicates, versions and languages.** Identify outdated copies, drafts and parallel versions of the same policy, and decide which one is authoritative. Note the languages of both documents and questions, because cross-language retrieval has to be tested explicitly.

3. Access control and privacy

Settle these before any real data leaves your systems.

- [ ] 10 **Permissions carried into retrieval.** If users may only see certain documents, retrieval must filter by the requesting user's permissions, synchronised from the source systems. Instructions in the prompt or trusting the model to withhold content is not access control.
- [ ] 11 **Personal data and legal basis.** Identify personal data in documents, questions and logs, and document the GDPR legal basis and purpose for processing it. Involve your data protection officer at the start, not at launch.
- [ ] 12 **Provider contract and data residency.** Put a data processing agreement in place with every model and infrastructure provider, review their sub-processors, and confirm that processing can stay in an EU region if you require it. Get written confirmation that your data is not used to train the provider's models.
- [ ] 13 **Retention, logging and redaction.** Decide how long prompts, retrieved passages and answers are stored, who can read them and whether personal data is redacted before logging. Logs are needed for debugging and evaluation, so the goal is controlled retention, not zero logging.

4. Retrieval design

Most wrong answers in RAG systems trace back to retrieval rather than the model; for a small, stable corpus, long context with prompt caching may replace retrieval entirely (see RAG vs fine-tuning).

- [ ] 14 **Chunking along document structure.** Split content by headings, sections, list items and table boundaries rather than fixed character counts. Keep the heading path with each chunk so that a passage still makes sense on its own.
- [ ] 15 **Metadata and filters.** Store title, source, section, date, language, version and access groups with every chunk. Metadata is what makes permission filtering, "latest version only" rules and useful citations possible.

- [ ] 16 **Hybrid search and reranking.** Combine keyword search for exact terms such as product codes, article numbers and names with vector search over embeddings for meaning, then rerank the combined candidates. Test each step against your example questions instead of assuming the defaults are good enough.
- [ ] 17 **Citations required.** Require the assistant to cite the passages it used, with a link to the source document and section. Citations let users verify answers and let reviewers see whether a wrong answer came from bad retrieval or bad generation.

5. Model and provider choice

Choose the model once you have an evaluation set, so the decision rests on your data rather than on generic benchmarks.

- [ ] 18 **Hosting option.** Compare a hosted API, the same or similar models in an EU cloud region, and a self-hosted open-weight model on your own infrastructure. Each option shifts the balance between answer quality, data control, operational effort and cost.
- [ ] 19 **Latency and cost per query.** Measure end-to-end response time and cost per answered question on your own example questions, including retrieval, reranking and input and output tokens. Prompt caching and smaller models for simple steps can change these figures considerably.
- [ ] 20 **Fallback and lock-in.** Plan what happens when the provider is down or retires a model: a second provider, a degraded mode or a clear error message. Keep prompts, the evaluation set and the retrieval layer provider-neutral, so that switching is a configuration change plus an eval run rather than a rewrite.

6. Evaluation

If you cannot measure quality, you cannot improve it or defend the go-live decision.

- [ ] 21 **Eval set before build.** Turn the real questions from section 1 into a versioned evaluation set with expected answers and expected sources, and extend it with hard and edge cases. Build it before the first prototype, not after the first demo.
- [ ] 22 **Retrieval and answer quality.** Measure separately whether the right passages were retrieved and whether the answer is correct, complete and faithful to those passages. Check citation accuracy: the cited passage must actually support the statement.
- [ ] 23 **Refusal and prompt-injection tests.** Include questions the assistant must decline and questions its sources cannot answer, and verify that it says so instead of guessing. Add documents and inputs that try to override its instructions or extract other users' data, and confirm that they fail.
- [ ] 24 **Regression runs and human review.** Run the full eval set on every change to prompts, models, chunking or data, and compare the results with the previous run. Automated grading with a model speeds this up, but a domain expert should spot-check results regularly.

7. Integration and operations

The assistant has to live somewhere, and someone has to run it after launch.

- [ ] 25 **Channel and sign-in.** Decide where the assistant lives: a website widget, Slack or Microsoft Teams, an internal tool or an existing ticket system. Use your existing SSO so that identity and permissions come from the same place as everywhere else.
- [ ] 26 **Human handoff and feedback.** Define how a user reaches a person when the assistant cannot help, with the conversation passed along. Add simple feedback buttons and route negative feedback into the evaluation set.
- [ ] 27 **Cost model and monitoring.** Estimate cost per query and per month at expected and peak usage, and set budget alerts. Monitor latency, error rates, refusal rates and user feedback, with logs redacted as agreed in section 3.
- [ ] 28 **Re-indexing, content ownership and incidents.** Automate re-indexing when source documents change, including deletions, so the assistant never quotes withdrawn content. Name who fixes content problems and define an incident process for wrong, harmful or leaked answers, including how to switch the assistant off quickly.

8. Go/no-go criteria

Agree on the decision rules before results come in, so the decision is not made on the strength of a good demo.

- [ ] 29 **Thresholds agreed in advance.** Write down minimum levels for answer correctness and citation accuracy on the eval set, acceptable latency and cost per query, and zero tolerance for answers that cross permission boundaries. Compare the results with the baseline from section 1.
- [ ] 30 **Time-boxed pilot with a clear exit.** Run a pilot with a small group of real users for a fixed period, with a named person who makes the call. Narrowing the scope or stopping is a legitimate outcome, and far cheaper than finding out after a full rollout.

How we can help

If you want to work through this checklist with us, the most direct route is our fixed-scope **AI Proof-of-Value Sprint**. In two weeks we scope the use case with you, build a RAG or agent prototype on your own data, create an evaluation set, prepare a cost model and deliver a go/no-go report you can take to your decision makers. Details are on our pricing page and in our AI & Automation services.

Before any of your documents reach a model, we agree with you how data is handled; you can read up front how we handle client data in AI projects. If you would rather start with a conversation, request a free estimate and describe the use case you have in mind.