

baswe.Ai Engineer Accelerator™

LLMs & RAG Checkpoint

Format:

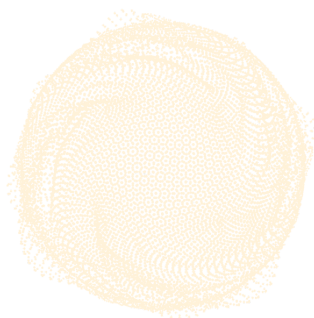
Build + Loom + Exam

Time investment:

Weekend sprint (8–12 hours)

Stack:

Python · LangChain · OpenAI · Pinecone



baswe's

AiEngineerAccelerator

Pre-checkpoint checklist

Before you start, confirm all of the following.

Accounts & API access

- OpenAI API key.
- Pinecone account, free tier is sufficient (Starter project + serverless index).
- GitHub account
- Loom account (free tier is fine for a single recording).

Local environment

- Python 3.10 or later.
- Git, configured with your name and email.
- A code editor with Python tooling
- Terminal comfort with venv / uv / poetry

Stack

- LangChain (latest stable)
- OpenAI — gpt-(recent)-mini for generation, text-embedding-3-small for embeddings (you may swap up to (recent) or text-embedding-3-large if you justify the cost in Part 2).
- Pinecone
- RAGAs or LangChain evaluators
- Pytest: for your test suite.

Part 1: The Build

Scenario

You have been hired as the first AI engineer at a 200-person SaaS company. The Customer Success team handles ~400 support tickets per day, and every senior CSM spends roughly 90 minutes per day searching internal documentation, runbooks, and post-mortems to answer questions they have answered before.

Your task: ship an internal RAG-powered support assistant that retrieves from the company's knowledge base and generates grounded, citation-backed answers. This is a v1 and does **not** need to be deployed to production, but must be evaluated like it will.

Required corpus

Use the provided corpus (link in LLMs & RAG -> Checkpoint -> START) — approximately 100 mixed-format documents (PDF, Markdown, HTML) covering product docs, internal runbooks, and resolved support tickets. The corpus includes:

- ~60 product documentation files (Markdown)
- ~25 internal runbooks (PDF, some scanned, some text-native)
- ~15 resolved support tickets with resolutions (HTML)

A test set of 50 evaluation queries with ground-truth answers and expected source documents is also provided. Your eval harness will score against this set.

Required architecture

Ingestion	Document loader → chunker → embedder → vector store. Chunking strategy must be justified in your README.
Retrieval	Top-k vector retrieval is the floor. You must implement at least one improvement: hybrid search (BM25 + vector), reranking (Cohere Rerank or similar), or query rewriting / multi-query.
Generation	LangChain chain or LCEL pipeline that produces a structured response: answer (string), sources (list of doc IDs), confidence (low / medium / high).
Evaluation	Eval harness that runs the 50-query test set and reports faithfulness, answer relevance, and context precision. Output must be machine-readable (JSON) AND human-readable (markdown summary).
Tests	pytest suite covering: ingestion runs without errors, retrieval returns ≥1 result for every test query, generation produces valid structured output, eval harness executes end-to-end.

Required repo structure

```
checkpoint-rag/
├── README.md           # Architecture decisions + how to run
├── pyproject.toml       # Or requirements.txt – pinned versions
├── .env.example        # Document required env vars (no real keys)
├── src/
│   ├── ingest.py       # Loads, chunks, embeds, indexes
│   ├── retrieve.py      # Retriever with your chosen improvement
│   ├── generate.py      # LCEL chain → structured response
│   └── pipeline.py      # End-to-end: query → answer
├── evals/
│   ├── harness.py      # Runs eval set, writes results
│   ├── test_set.json   # The 50 provided queries
│   └── results/        # Output directory (committed empty)
```

```
├── tests/
│   ├── test_ingest.py
│   ├── test_retrieve.py
│   ├── test_generate.py
│   └── test_evals.py
└── notebooks/
    └── exploration.ipynb      # Optional - your scratch work
```

Required technical checks

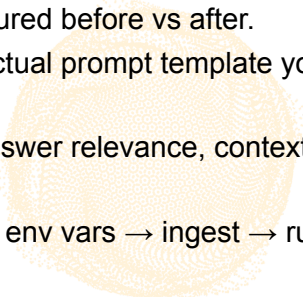
Your checkpoint should demonstrate all five of these.

1. `python -m pytest tests/` exits 0.
2. `python -m src.ingest --corpus ./corpus` runs end-to-end without unhandled exceptions.
3. `python -m evals.harness` writes a results JSON to `evals/results/`.
4. Faithfulness score on the eval set is ≥ 0.70 .
5. Context precision on the eval set is ≥ 0.60 .

README requirements

Your README should contain:

- One-paragraph summary of what you built and what it does.
- Architecture diagram (Mermaid, draw.io export, or a clear ASCII diagram is fine).
- Chunking strategy section: what size, what overlap, how you handle PDFs vs Markdown vs HTML, and why.
- Retrieval strategy section: which improvement you implemented (hybrid / rerank / multi-query), why you chose it, what you measured before vs after.
- Generation prompt: include the actual prompt template you used, with reasoning for the structure.
- Eval results table: faithfulness, answer relevance, context precision, plus 3 representative failure cases with your analysis.
- How to run: clone → install → set env vars → ingest → run query → run evals. Five commands or fewer.
- What you'd do with another week: ranked list of next improvements with the reasoning.

basw s

AiEngineerAccelerator

Part 2: Loom Walkthrough

Format

- Single Loom recording, 10–30 minutes, screen share with face cam on.
- Public or unlisted Loom link, included in your README

1. Architecture overview (4–6 minutes)

- Walk through your architecture diagram.
- Explain ingestion → retrieval → generation, end to end.
- Call out where you made a non-default decision and why.

2. Live demo (5–8 minutes)

- Run 3 queries against your system, live.
- Pick: one easy query (system handles cleanly), one medium (forces retrieval to work), one hard (likely to expose a weakness).
- After each, narrate what happened: did it retrieve the right context? Was the generation grounded?

3. Eval results & analysis (4–6 minutes)

- Show your eval harness output.
- Pick 2 failure cases and walk through what went wrong: was it retrieval, generation, or evaluation noise?
- Explain one specific change you would make and how you would measure the impact.

4. Tradeoffs you made (4–6 minutes)

- Walk through 3 specific tradeoffs you made under the time constraint.
- For each: what you chose, what you sacrificed, and what would change with another week.
- This is where engineering judgment shows. Generic answers ('I would add monitoring') score low. Specific answers ('I would add per-query latency tracking with p95 alerting at 3s because right now I have no signal on retrieval slowdowns at scale') score high. **Companies love engineers with opinions, it shows proof of tested beliefs.**

5. What you would do differently (3–5 minutes)

- Honest reflection: what would you change if you started over?
- What did you not know going in that you know now?

baswe's

AiEngineerAccelerator

Part 3: Technical Exam

Format

- Open-book. Open-internet.
- Do **NOT** use an LLM to generate your answers. Do **NOT** Google the entire question. Cheating on these only cheats yourself. Use of LLMs to draft answers is grounds for **failing the entire checkpoint**.
- Time-boxed: 1-2 hours. We do not enforce the timer, but if you spend 6+ hours on this section, that is a signal you might not have understood the concepts deep enough.
- Written answers in a single Markdown file (exam-answers.md)

Section A: Foundations

Q1 (3 pts)

Explain the relationship between context window, tokens, and cost. Why does chunking strategy materially impact both retrieval quality AND inference cost? Use specific numbers from a model you have actually used.

Q2 (3 pts)

Compare temperature, top_p, and top_k. When would you use each, and why might you set temperature=0 in a production RAG system? When would temperature=0 be the wrong call?

Section B: Provider Judgment

Q3 (4 pts)

You're building an internal HR document Q&A system for a 5,000-person healthcare company. Compare OpenAI GPT-(recent), Claude Sonnet, and self-hosted Llama 3 across (a) cost at scale, (b) data privacy and compliance, (c) latency, (d) accuracy on long-context retrieval. Pick one. Defend the choice in 4–6 sentences.

Q4 (2 pts)

Give two specific scenarios where fine-tuning Llama 3 is the correct choice over prompt engineering with GPT-(recent). What is the use case, what does the training data look like, and what does success measurement look like?

Section C: RAG Architecture

Q5 (3 pts)

You have 50,000 PDFs of insurance policies, each 30–100 pages, with tables, headers, footers, and inline citations. Walk through your chunking strategy: chunk size, overlap, how you handle tables, what metadata you attach. Justify each decision.

Q6 (2 pts)

AiEngineerAccelerator

Explain hybrid search (BM25 + semantic). When does it meaningfully outperform pure vector search? Give a concrete example with the kind of query that would fail under pure vector retrieval but succeed under hybrid.

Q7 (2 pts)

What is reranking, and why do production RAG systems often add a reranker even when retrieval looks 'good enough' on top-5 metrics? Name a specific reranker you would use and explain when you would reach for it.

Q8 (3 pts)

Compare vector RAG vs Graph RAG. Describe a use case where Graph RAG meaningfully outperforms vector RAG and explain the underlying reason. **Not** just 'it has relationships' but specifically what kind of query benefits from a traversal that embeddings cannot answer.

Section D: Evaluation & Production

Q9 (3 pts)

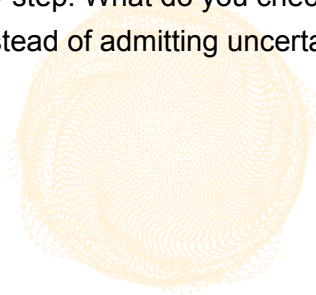
Define faithfulness, answer relevance, and context precision in RAG evaluation. For each, give one concrete failure mode you have seen (or could see) in your build from Part 1. How would you collect a ground-truth set if you didn't have one provided?

Q10 (3 pts)

List 3 production failure modes for a deployed RAG system and how you would detect each in monitoring. Be specific about the signal, not 'monitor accuracy' but what metric, what threshold, what alert.

Q11 (2 pts)

Your RAG chatbot is hallucinating on questions where the answer genuinely is not in the corpus. Walk through your debugging process step by step. What do you check first, second, third? What is the fix when the issue is the model 'helping' instead of admitting uncertainty?



baswe's

AiEngineerAccelerator