Day 3 – Research vs Reasoning: RAG Systems Portfolio

Topic: RAG systems: when they help, when they don't
Duration: 60 minutes
Date: [Session Date]

Overview

This session explored Retrieval-Augmented Generation (RAG) systems through three distinct lenses: architectural frameworks, failure mode analysis, and practical explanation. The goal was to understand not just how RAG works, but when it works and when it becomes a liability.

Step 1: Perplexity – RAG Architecture Examples

Key Findings

RAG systems integrate retrieval of external knowledge with language model generation to ground responses in relevant data. They prove valuable for tasks requiring up-to-date or proprietary information but underperform in creative reasoning or when retrieval fails.

When RAG Helps

- **Proprietary data:** Internal documentation, policies, knowledge bases
- **Rapidly changing knowledge:** News, financial data, market conditions
- **Traceability requirements:** Healthcare, finance, legal—fields requiring auditable sources
- **Minimizing hallucinations:** Anchoring outputs to retrieved sources
- **Selective invocation:** Using RAG only for precision-dependent queries optimizes performance

When RAG Doesn't Help

- **Ideation and creative tasks:** Where retrieval constrains rather than aids generation
- **Simple summarization:** Static knowledge already learned during training
- **Multi-hop reasoning:** Questions requiring synthesis across multiple documents
- **Chunking failures:** Poor document segmentation leads to incomplete or incoherent context
- **Retrieval noise:** Irrelevant results overwhelm the prompt, adding latency and cost

Architecture Examples: Perplexity's Approach

Perplexity's RAG pipeline features:

- Intelligent web crawling
- Vector indexing of retrieved content
- Similarity-based retrieval
- Dynamic augmentation into multi-LLM prompts (GPT-4, Claude)
- Cited generation with source attribution
- Hybrid semantic-keyword search with re-ranking
- User feedback loops to refine retrieval quality

Other variants include modular platforms like CLŌD, which toggle RAG conditionally for production control.

Step 2: Claude – Failure Modes Analysis

Core Failure Categories

1. Retrieval Quality Issues

Poor chunking fragments documents mid-concept. Irrelevant retrievals occur when similarity metrics don't align with semantic relevance. Knowledge base pollution (outdated, duplicated, low-quality information) degrades output regardless of LLM capability.

2. Context Overload & Degradation

Prompt overload occurs when retrieved context exceeds effective working window. The LLM processes irrelevant chunks, crowding out reasoning space. More irrelevant text degrades performance even with long-context models.

3. Multi-Hop Reasoning Breakdown

RAG struggles with questions requiring synthesis across multiple documents. Each retrieval may succeed individually, but subsequent retrievals may miss context from earlier steps. The system becomes brittle without iterative refinement.

4. Latency & Cost Overhead

For tasks with sufficient knowledge in the base model, RAG adds retrieval, embedding, re-ranking, and integration overhead without benefit. Simple queries become slower and more expensive.

5. Task-Misalignment

Creative ideation, brainstorming, and open-ended synthesis don't benefit from retrieval. Summarization and simple transformations become constrained by enforced grounding.

Why Failures Occur

- Treating RAG as universally beneficial rather than task-specific
- Building on incomplete or outdated knowledge bases
- Using naive chunking strategies without semantic awareness
- Assuming single-pass retrieval is sufficient
- Absence of feedback loops to refine retrieval quality

Mitigation Strategies

- Selective invocation: route simple queries away from RAG
- Adaptive chunking with semantic awareness
- Re-ranking and filtering before prompt injection
- Iterative feedback loops to flag irrelevant retrievals
- Continuous refinement of the knowledge base

Step 3: Plain-English Explanation

RAG Without Hype

RAG (Retrieval-Augmented Generation) is a way to help a language model look things up before it speaks.

How it normally works: An LLM answers from what it learned during training. That works for general knowledge, but it fails when the information is private (company docs, internal policies), new (today's data), or sensitive enough that guessing is dangerous.

How RAG fixes it:

1. **Retrieval:** The system finds relevant documents from an external source—usually a vector database built from PDFs, webpages, or internal files. Retrieval is based on semantic similarity, not exact keywords.
2. **Augmentation:** The retrieved text is inserted into the model's prompt as context.
3. **Generation:** The LLM generates an answer grounded in that context, instead of relying purely on memory.

Why it helps:

- The model can answer questions about data it never saw during training

- Answers can be traced back to real documents
- Hallucinations are reduced because the model is anchored to sources

Why it's not magic:

- If retrieval is bad, generation will be bad
- If the task doesn't need outside information, RAG adds noise and cost
- The model still reasons—RAG only supplies raw material

The right mental model: RAG doesn't make the model smarter. It makes the model better informed at the moment of answering. Used selectively, it's powerful. Used everywhere, it becomes expensive baggage.

Reflection: Which Tool Mixed Facts with Assumptions?

Analysis

Perplexity's output presented concrete architectural examples and documented use cases. However, it made subtle claims about what "works" without deep examination of conditions. Statements like "RAG excels with proprietary data" are true but incomplete—they gloss over the fact that RAG can still fail spectacularly with poor chunking or retrieval quality. The architecture examples are factual but presented as solutions without acknowledging failure modes.

Claude's analysis deliberately foregrounded failure modes and explicitly named the gap between marketing claims and reality. It provided mechanistic explanations (why chunking fails, why multi-hop reasoning breaks) rather than assertions. The tone was analytical rather than prescriptive.

The plain-English explanation intentionally separated hype from function. Its framing ("RAG doesn't make the model smarter") directly contradicts marketing language while staying factual. It treats RAG as a tool with clear trade-offs rather than a solution.

The Mix

Perplexity mixed facts with assumptions most noticeably. Phrases like "RAG excels," "minimizes hallucinations," and "enables auditable responses" are presented as facts but rest on unstated assumptions:

- That retrieval will succeed
- That chunking is well-designed
- That the knowledge base is current and clean
- That the task actually benefits from external grounding

These are conditional claims presented as unconditional ones.

Claude's approach avoided this by explicitly naming conditions ("RAG falters," "when retrieval fails," "poor chunking"). It treated assumptions as hypotheses to examine rather than facts to assert.

The plain-English explanation avoided the mix entirely by being explicit about constraints ("if retrieval is bad," "if the task doesn't need outside information").

Key Learning

The difference isn't between tools—it's between claiming truth and examining conditions. RAG is neither universally good nor universally bad. Architectural presentations naturally emphasize the positive use case; analysis naturally emphasizes failure points; plain-language explanation works by being honest about trade-offs.

A mature understanding of RAG requires holding all three perspectives simultaneously.