FINE-TUNING vs RAG DECISION FRAMEWORK

Quick Reference for Stephen Conversation

THE FUNDAMENTAL DIFFERENCE

Fine-Tuning

"Change HOW the model thinks"

- Retrains model weights on custom dataset
- Modifies the model's behavior/style/capabilities
- The model LEARNS new patterns

RAG (Retrieval-Augmented Generation)

"Give the model WHAT to think about"

- Fetches relevant context and adds it to the prompt
- The model stays the same, but gets better information
- No retraining needed

YOUR MISCONCEPTION (FIXED)

❌ **What you thought fine-tuning was:** "Feedback loop in prompts" or "few-shot learning"

✅ **What fine-tuning actually is:** Retraining model weights using gradient descent on a custom dataset

- Takes hours/days
- Requires training infrastructure
- Updates the model's parameters permanently
- Like teaching someone a new skill vs giving them a reference book

---

THE THREE TECHNIQUES COMPARED

Technique	What Changes	Speed	Cost	Use Case
Prompt Engineering	Nothing (just the prompt)	Instant	$	Quick iteration, testing
Few-Shot Learning	Nothing (examples in prompt)	Instant	$$	Show model the pattern
RAG	Nothing (adds context to prompt)	Fast (ms)	$$	Dynamic knowledge, fresh data
Fine-Tuning	Model weights	Slow (hours)	$$$$	Specialized behavior, style

---

WHEN TO USE RAG

✅ **Use RAG when you need:**

1. **Dynamic/Changing Information**
    - Company docs that update weekly
    - Real-time data (stock prices, news)
    - User-specific information
    - **Your nonprofit case:** Mission statements change, new nonprofits added
2. **Transparency/Citations**
    - Need to show sources
    - Compliance/audit requirements
    - "How did you know this?"
3. **Large Knowledge Base**
    - More context than fits in prompt
    - Millions of documents
    - Only need relevant subset
4. **Cost Efficiency**
    - Cheaper than fine-tuning
    - No retraining needed
    - Update knowledge base, not model

RAG Architecture:

User Query
↓
Convert to embedding
↓
Search vector database (cosine similarity)
↓
Retrieve top K relevant documents
↓
Augment prompt: [Query + Retrieved Context]
↓
Send to LLM
↓
Response (grounded in retrieved docs)

WHEN TO USE FINE-TUNING

✅ **Use Fine-Tuning when you need:**

1. **Specialized Behavior**
    - Model needs to write in specific style
    - Follow complex domain-specific rules
    - Generate structured outputs consistently
    - Example: Legal document generation, medical coding
2. **Consistent Pattern**
    - The knowledge is stable (doesn't change often)

    - Same format needed repeatedly
    - Example: Always format SQL queries a certain way
3. **Reduce Latency**
    - Model knows patterns internally (no retrieval step)
    - Faster than RAG for high-frequency requests
4. **Small, Specialized Domain**
    - Limited vocabulary/concepts
    - Deep expertise in narrow area
    - Example: Customer service for specific product

Fine-Tuning Process:

1. Collect training data (examples of input → desired output)
2. Format as training dataset
3. Run training job (updates model weights)
4. Validate performance
5. Deploy fine-tuned model

Cost: $$$$ (GPU time, engineering effort)
Time: Hours to days
Benefit: Model "knows" the patterns internally

YOUR NONPROFIT USE CASE ANALYZED

Mission Comparison Feature

Problem: Compare similarity of nonprofit mission statements

Option A: What You Built (LLM API Calls)

- Send each pair to LLM for comparison
- Cost: $2 per 200 comparisons
- Time: 2 minutes
- Problem: Expensive, slow, not scalable

Option B: Embeddings (BEST) ✅ Winner for your use case

- Create embeddings for all missions
- Use cosine similarity
- Cost: $0.02 per 200 missions
- Time: 5 seconds
- Why: This is a similarity problem, not a reasoning problem

Option C: RAG

- When: If you need to explain WHY they're similar
- Architecture:

1. User asks: "Find missions similar to X"
2. Embed X, search vector DB
3. Retrieve top 10 similar missions
4. RAG prompt: "Here are similar missions: [...]. Explain the commonalities."
5. LLM generates explanation

- Cost: Moderate
- Why useful: Gives qualitative insight, not just numbers

Option D: Fine-Tuning ❌ Wrong tool for this problem

- When: If you needed the model to consistently format mission comparisons in a specific proprietary way
- Why not: The problem is similarity search, not teaching new behavior
- Cost: $1000s
- Why wrong: Massive overkill, knowledge changes (new nonprofits added)

DECISION TREE

START: What's your goal?

├── Need to compare/search similar items?
│ └── → USE EMBEDDINGS + COSINE SIMILARITY
│
├── Need fresh/dynamic knowledge?
│ ├── Data changes frequently?
│ │ └── → USE RAG
│ └── Need to cite sources?
│ └── → USE RAG
│
├── Need consistent specialized behavior?
│ ├── Pattern is stable?
│ │ └── → CONSIDER FINE-TUNING
│ └── Need specific style/format?
│ └── → CONSIDER FINE-TUNING
│
└── Just need better prompts?
└── → USE PROMPT ENGINEERING / FEW-SHOT

COMBINING TECHNIQUES

Real-world systems often combine approaches:

Example: Customer Support Bot

1. **Embeddings:** Find similar past tickets
2. **RAG:** Retrieve relevant documentation
3. **Fine-Tuning:** Respond in company's brand voice
4. **Prompt Engineering:** Structure the final response

Your Nonprofit Feature (Enhanced)

1. **Embeddings:** Fast similarity search
2. **RAG:** Fetch mission details + program info
3. **Prompt Engineering:** "Explain in 2 sentences why these are similar"
4. **NOT fine-tuning:** Data changes too often, similarity is the core task

---

COMMON MISTAKES

Mistake 1: "Let's fine-tune for everything!"

- Reality: Expensive, slow, often unnecessary
- First try: Better prompts, then RAG, then consider fine-tuning

Mistake 2: "RAG will solve all hallucination"

- Reality: Only if retrieved docs have the answer
- RAG reduces hallucination but doesn't eliminate it

Mistake 3: "Embeddings are fine-tuning"

- Reality: Completely different. Embeddings = representation. Fine-tuning = retraining.

Mistake 4: "We can't use RAG because our data is proprietary"

- Reality: RAG works great with private data. You control the vector database.

COST COMPARISON (Real Numbers)

Your 200 nonprofit mission comparisons:

Approach	Setup Cost	Per-Query Cost	Total Cost	Speed
LLM API calls	$0	$0.01	$2.00	2 min
Embeddings	$0.02	~$0	$0.02	5 sec
RAG (with explanation)	$0.02	$0.001	$0.22	10 sec
Fine-Tuning	$500+	$0.01	$502+	2 min*

*After expensive training process

KEY TERMS FOR STEPHEN

Vector Database:

- Specialized database optimized for similarity search
- Stores embeddings
- Examples: Pinecone, Weaviate, Chroma, FAISS
- Returns nearest neighbors by cosine similarity

Retrieval:

- The "R" in RAG
- Fetching relevant documents from knowledge base
- Uses embeddings + similarity search

Augmentation:

- The "A" in RAG
- Adding retrieved context to the prompt

Generation:

- The "G" in RAG
- LLM produces response grounded in retrieved context

Few-Shot Learning:

- Providing examples in the prompt (not fine-tuning!)
- "Here are 3 examples of the format I want..."

STEPHEN'S LIKELY FOLLOW-UP QUESTIONS

Q: "When would you use fine-tuning over RAG?"
A: "When I need consistent specialized behavior that's stable over time. For example, if we needed the model to always format nonprofit analysis reports in a specific proprietary structure, fine-tuning makes sense. But for our mission comparison feature, the knowledge changes frequently and it's fundamentally a similarity problem, so embeddings + RAG is the right architecture."

Q: "What about hybrid approaches?"
A: "Absolutely. In production I'd likely use embeddings for similarity search, RAG to pull in additional context like program details or impact metrics, and prompt engineering to format the explanation. I wouldn't fine-tune because the data changes too frequently and the core task is similarity measurement, not specialized reasoning."

Q: "How do you evaluate if RAG is working?"
A: "Three key metrics: (1) Retrieval quality - are we fetching relevant docs? (2) Answer quality - is the generated response accurate? (3) Citation accuracy - can we trace back to source. I'd build a test set of queries with known good answers and measure these systematically."

BOTTOM LINE FOR YOUR USE CASE

Mission Comparison Feature:

1. **First:** Embeddings for similarity (fastest, cheapest, most appropriate)
2. **Then:** RAG if users want explanations (add value with context)
3. **Never:** Fine-tuning (wrong tool, data changes, expensive)

Stephen wants to see you understand: Matching the technique to the problem type

READ OVER SHABBAT: Chapter 6 of Fine-tuning guide + RAG article to deepen these concepts