Personalizing LLMs

Fine-Tuning and Practical Alternatives

Dario G. Flores
Grid Dynamics

Pythonistas GDL

Why Do We Need LLM Personalization?

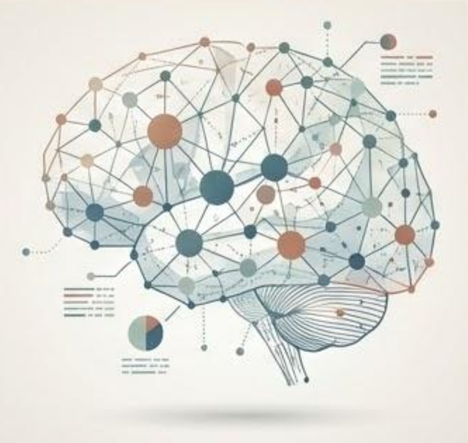
Base LLM (Out of the Box)

- Knows everything
- Optimized for conversation
- Inconsistent formats
- No domain context

Reality check

Production Systems Need

- Predictable outputs
- Domain-specific language
- Structured data (JSON, schemas)
- Low latency & controlled cost



Problem	Technique
Explore / prototype	Prompting
External knowledge	RAG
Actions & workflows	Tools
Consistent behavior	Fine-Tuning

Pythonistas GDL
Pythonistas Community

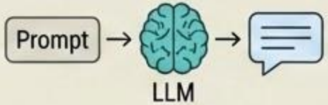
The Personalization Spectrum

LLM personalization is a spectrum — not a single technique.

As you move across the spectrum, you gain control and consistency, but pay more in cost and complexity.

Prompt Engineering (Zero-Code Personalization)

What it is?



- Shaping model behavior using instructions and examples
- No model changes, just better prompts

How it works?

- System prompts
- Few-shot examples
- Constraints (“answer as...”, “format as...”)

Python examples

Role-based: prompt = """
You are a senior Python backend engineer.

Few-shot:
```
prompt = """Translate to Spanish.
English: Hello
Spanish: Hola
English: Goodbye
Spanish: Adios
English: Good morning"""
```

Constraint:
```
prompt = """Classify the sentiment.
Answer as a JSON object with
'sentiment' and 'score'.
Text: I love this!"""
```

Pros:

- Fast
- Cheap
- No infrastructure

Limits:

- No memory
- No real domain knowledge
- Fragile with complex logic

Use when:

- Prototyping
- Simple role-based behavior
- Demos & experiments

Pythonistas GDL
Pythonistas Community

Zero-Shot Prompting

What is it?

Ask the model to do a task without examples

Example

Classify this support ticket as:
Billing, Technical, or General.
Ticket: "My payment failed twice."

→ [Model Output]
Billing

Characteristics

- Relies on the model's pretraining
- Fast
- Minimal tokens

When it works well

- Common tasks
- Clear instructions
- Well-known domains

Few-Shot Prompting

What is it?

Give the model a few examples of the task

Why it works

- Anchors behavior
- Reduces ambiguity
- Improves consistency

Tradeoff

- More tokens
- Still fragile at scale

Example

Ticket: “I forgot my password”
Category: Technical
Ticket: “I was charged twice”
Category: Billing
Ticket: “My payment failed twice”
Category:

→ [Model Output]
Billing

Why Few-Shot Exists at All

Few-shot prompting is a bridge technique:

- Too complex for zero-shot
- Too expensive to fine-tune
- It’s “temporary learning through context.”

Chain-of-Thought (CoT) Prompting

What is it?

Ask the model to explain its reasoning step-by-step before giving the final answer.

“Think step-by-step” is the classic example.

Example

Q: If I have 3 apples and buy 5 more, then eat 2, how many do I have?
A: Start with 3. Buy 5 → 3+5=8. ←
Eat 2 → 8-2=6. The answer is 6.

Characteristics

- More tokens (expensive)
- Slower latency
- Improves performance on complex tasks

When it works well

- Math/Logic word problems
- Multi-step reasoning
- Debugging code

Chain-of-Thought (CoT) Prompting

The Problem (Looks easy... but often fails)

Question

A service retries a failed request up to 3 times. Each retry has a 20% chance of success, independent of the others.

What is the probability that the request eventually succeeds?

This is perfect because:

- It's not trivial
- Many models answer it wrong without reasoning
- CoT fixes it immediately.

Step 1 — Naive Prompt (Often Wrong)

Prompt

A service retries a failed request up to 3 times.

Each retry has a 20% chance of success.

What is the probability that the request eventually succeeds?

Typical Wrong Answer

60%

- Model adds probabilities
- No reasoning
- Sounds confident, but wrong...

Step 2 — Chain-of-Thought Prompt (Same Model)

Prompt

Think through the problem step by step.
Then provide only the final numeric probability.

A service retries a failed request up to 3 times.
Each retry has a 20% chance of success.

Correct Answer

48.8%

Explanation:
Failure probability per try: 80%

Failure after 3 tries:
$0.8 \times 0.8 \times 0.8 = 0.512$

Success probability:
$1 - 0.512 = 0.488$ (48.8%)

The model didn't get smarter.
We just forced it to reason instead of guessing.

Retrieval-Augmented Generation (RAG)

What is it?

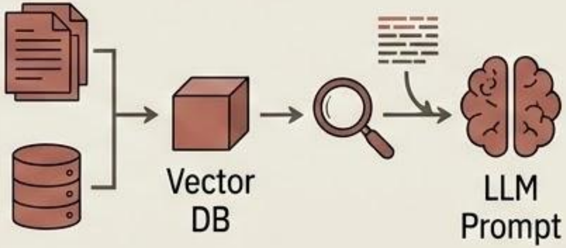
Combine LLMs + your data (documents, DBs, APIs)

How it works?

- Embed documents
- Store in vector DB
- Retrieve relevant chunks
- Inject into prompt

Python stack:

- sentence-transformers / OpenAI embeddings
- FAISS / Chroma / Pinecone
- LangChain / LlamaIndex



Pros:

- Up-to-date knowledge
- No retraining
- Scales well

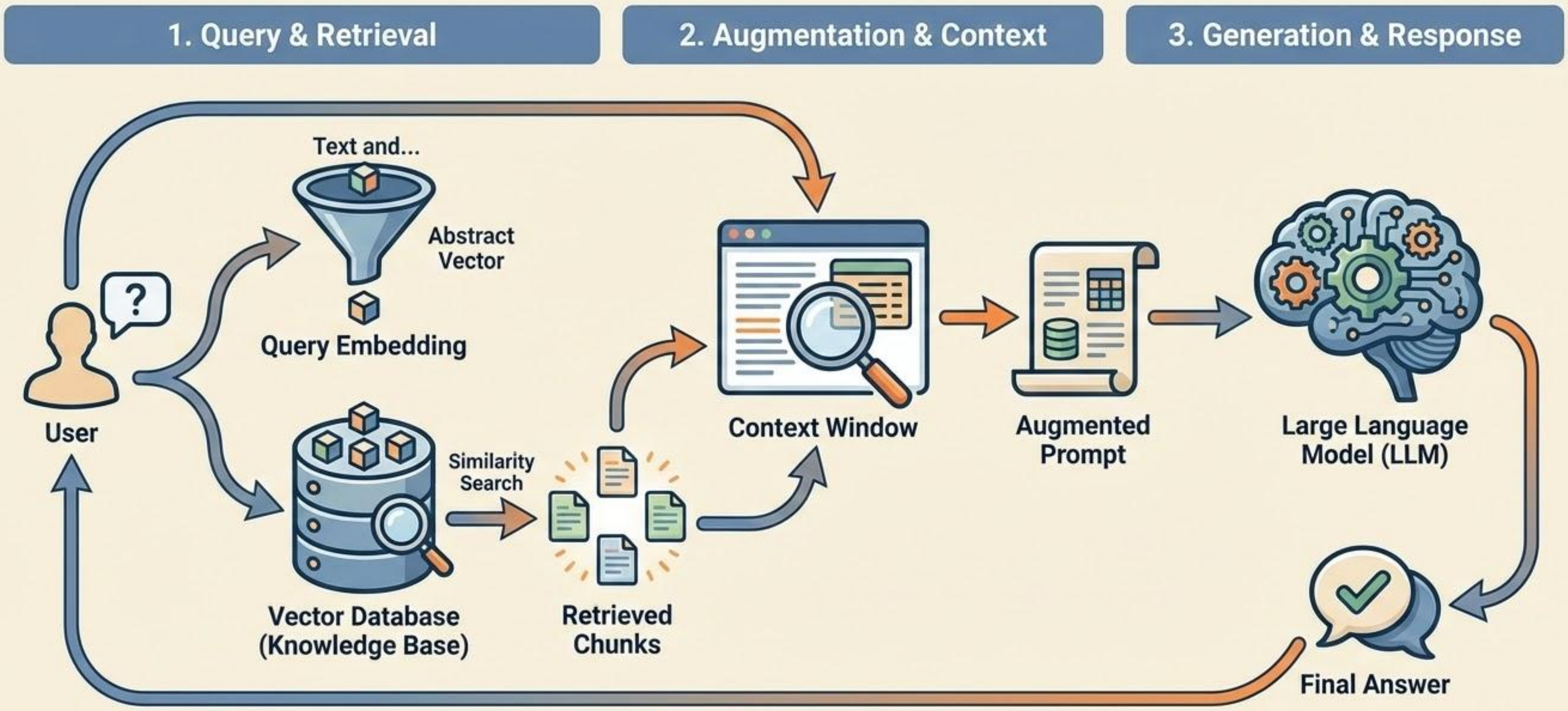
Limits:

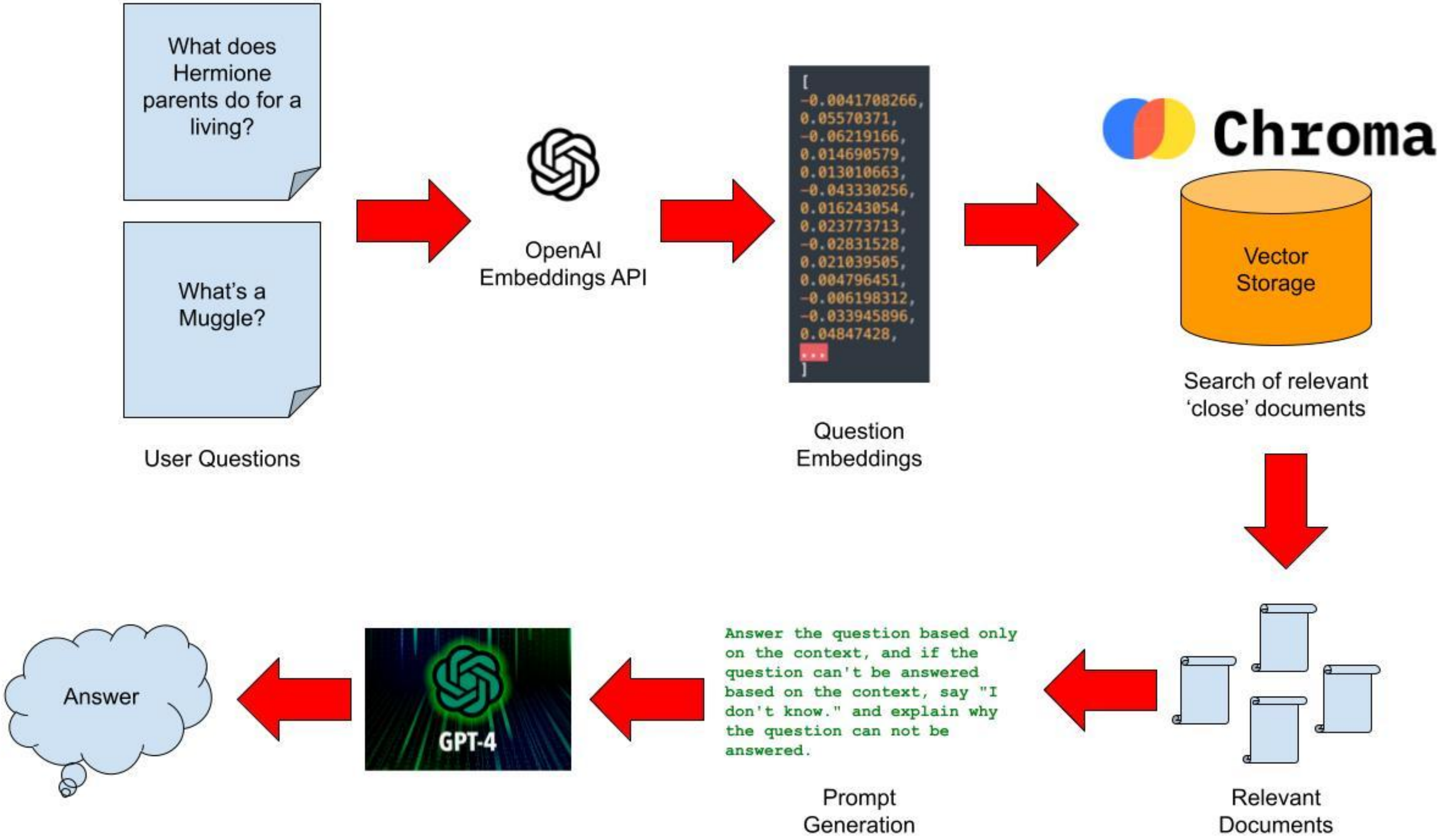
- Retrieval quality matters
- Context window limits
- More moving parts

Use when:

- Knowledge-heavy systems
- Company docs, manuals, wikis
- Search + QA bots

The RAG (Retrieval-Augmented Generation) Process Explained




What does Hermione parents do for a living?
What's a Muggle?
User Questions
OpenAI Embeddings API
[
-0.0041708266,
0.05570371,
-0.06219166,
0.014690579,
0.013010663,
-0.043330256,
0.016243054,
0.023773713,
-0.02831528,
0.021039505,
0.004796451,
-0.006198312,
-0.033945896,
0.04847428,
...
]
Question Embeddings
Chroma
Vector Storage
Search of relevant 'close' documents
Relevant Documents
Answer the question based only on the context, and if the question can't be answered based on the context, say "I don't know." and explain why the question can not be answered.
Prompt Generation
GPT-4
Answer

Tool / Function Calling (Behavioral Personalization)

What is it?

Let the model decide when to call code

Analogy:

🧠 LLM = brain

🙌 Python functions = hands

Example

```
def get_weather(city: str) -> dict:…

def get_order_status(order_id: str)
    -> dict: …

def create_support_ticket(user_id:
    str, issue: str) -> dict: …

def check_cpu_usage() -> float: …

def send_alert(message: str) ->
    None: …
```

LLM decides:

"I need real-time data → call function"

Pros:

- Deterministic actions
- Real-world interaction
- Safe execution boundaries

Limits:

- Still not learning
- Requires orchestration logic

Use when:

- Agents
- Automation
- Data pipelines
- Backend workflows

From Tool Calling to Agentic RAG

Tool / Function Calling

- LLM decides when to call a function
- Deterministic, single-step actions
- Acts on known needs

But most real problems require…

Agentic RAG

- Multi-step reasoning
- Dynamic information retrieval
- Planning + tools + memory

What Changes with Agentic RAG?

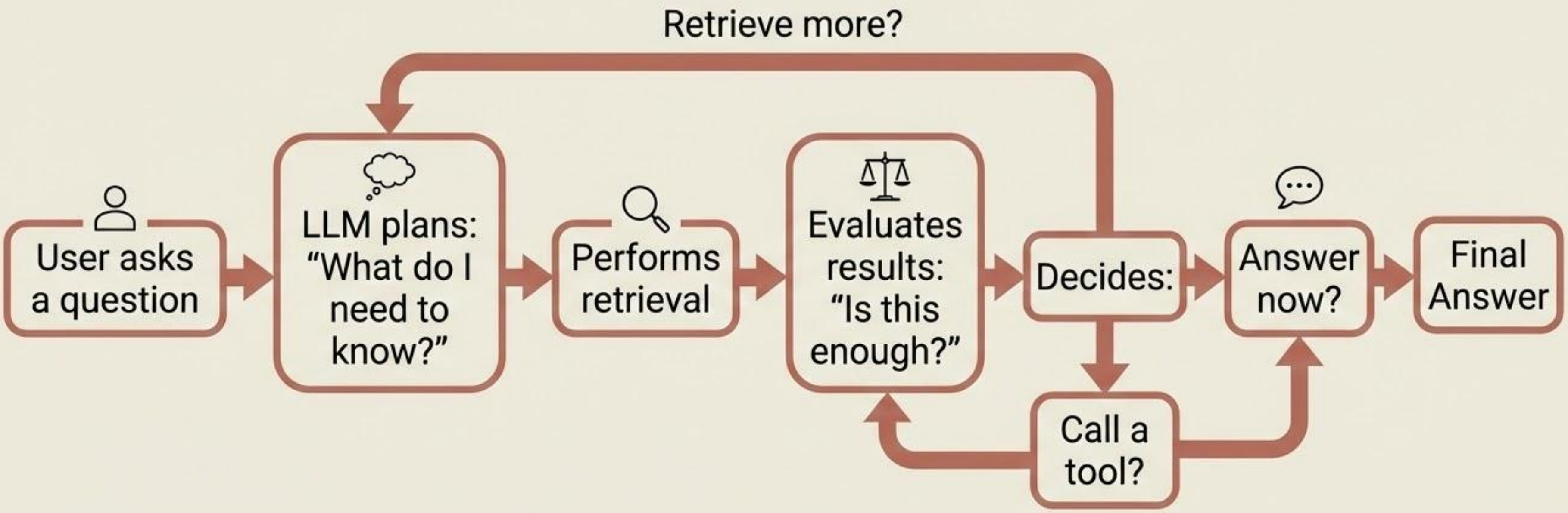
Traditional RAG

- Retrieve → Generate
- One retrieval step
- Passive

Agentic RAG

- Reason → Retrieve → Evaluate → Repeat
- Chooses what to retrieve next
- Stops when confident

How Agentic RAG Works (Step-by-Step)



Retrieval becomes iterative and goal-driven.

Pythonistas GDL
Pythonistas Community

Agentic RAG Use Case

User asks

"Why did order #78421 fail delivery?"

Agentic RAG behavior

1. Reason
- Identify missing information
- Break question into sub-queries

2. Retrieve
- Order details (OMS / DB)
- Carrier delivery logs
- Customer history

3. Evaluate
- Check consistency & confidence
- Decide if more data is needed

4. Act
- Escalate if required
- Call create_support_ticket()

iterative

Key properties

- Retrieval is dynamic, not fixed
- Tools are called only when needed
- Loop stops when confidence is high

The agent controls the investigation — not you.

Python Tools & Frameworks for Agentic RAG

Orchestration & Agents

- LangChain
- LangGraph
- LlamaIndex (Agents)
- CrewAI

Retrieval & Vector DBs

- LlamaIndex (RAG)
- Pinecone
- Weaviate
- ChromaDB

LLMs & Inference

- OpenAI API
- Anthropic API
- Hugging Face
- ollama (local)

Tools & APIs

- LangChain Tools
- Serper (Search)
- Zapier NLA

Fine-Tuning

This is where things get serious.

What Is Fine-Tuning?

Fine-tuning is training an existing **LLM** on your own examples to change how it behaves.

Key clarifications:

1. You are **not** training from scratch
2. You are **not** adding new knowledge
3. You are adjusting the model's weights so it:
   - -) **Follows** instructions more reliably
   - -) Produces consistent outputs
   - -) Adopts a specific style or structure

Prompting asks politely. | Fine-tuning rewires habits.

Supervised Fine-Tuning (SFT)

What SFT Really Is

Supervised Fine-Tuning **= teaching the model by example.**

You give the model:

- An input,
- The exact output you want,
- And you repeat this hundreds or thousands of times.

The model learns:

- What to focus on,
- What to ignore,
- How to respond consistently.

What Changes After SFT

Before SFT:

- Long prompts
- "Please output JSON"
- Format breaks
- Inconsistent tone

After SFT:

- Short prompts
- Stable structure
- Predictable behavior
- Lower latency

The model stops guessing what you want... **now it knows.**

Example – Structured Log Analysis

Input:

```
ERROR 2025-01-18 12:01:22
Connection timed out while accessing
inventory-service
```

Output:

```
{
  "error_type": "TIMEOUT",
  "service": "inventory-service",
  "severity": "HIGH",
  "recommended_action": "Check
connection  pool and network latency"
}
```

Why SFT Works Here?

- Repetitive pattern
- Fixed schema
- Clear expectations

This is repetitive work with a non-ambiguous output and that’s exactly why SFT is perfect.

Example – Customer Support Classification

Input:

```
My order arrived damaged and I want a
refund.
```

Output:

```
{
  "category": "DAMAGED_ITEM",
  "sentiment": "NEGATIVE",
  "priority": "HIGH"
}
```

Why Not Prompting?

- Thousands of similar tickets
- Output must be identical every time
- Prompting drifts*

Humans do it by pattern recognition, SFT beats prompts and teaches this behaviour to the LLM.

* Prompt drift is the phenomenon where a LLM produces increasingly inconsistent, irrelevant, or degraded outputs over time, even when the input prompt remains identical.

Example — Code Review Assistant

Input:

```
def add(a, b):
    return a + b
```

Output:

```
{
  "issues": [],
  "suggestions": [
    "Add type hints for better readability",
    "Add docstring"
  ],
  "severity": "LOW"
}
```

Why This Is Powerful

- Style consistency
- Same review tone
- Same structure every time

What Makes a GOOD SFT Dataset

✅ High-Quality Signals

- Same structure everywhere
- Clear intent
- No ambiguity
- No explanations, just results

❌ Bad Signals

- Mixed formats
- Inconsistent keys
- Verbose prose
- Contradictory labels

The model is optimizing for likelihood. It’s rewarded for reproducing patterns it saw. It has no concept of ‘this was a mistake’. It averages over **behavior**, including the bad parts.

Reinforcement Learning from Human Feedback (RLHF)

What is it?

- Humans rank model outputs
- Model learns preferences via reinforcement learning

What it optimizes:

- Helpfulness
- Safety
- Tone
- Alignment

Reality check:

- Expensive
- Hard to run yourself
- Mostly done by large labs

RLHF doesn’t teach facts, it teaches manners.

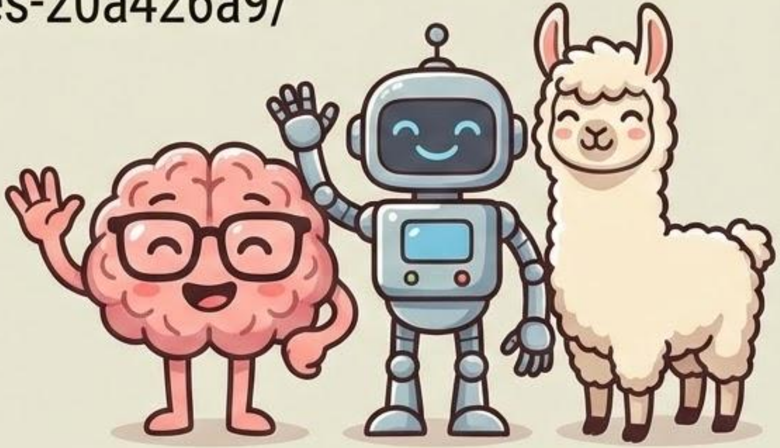
Comparison Table

Technique	Changes Behavior	Uses External Data	Cost	Best For
Prompting	❌	❌	$	Prototypes
RAG	❌	✅	$$	Knowledge
Agentic RAG	❌ *(emergent)*	✅ *(multi-source)*	$$$	Investigations & workflows
Tools	❌	✅	$$	Actions
Fine-Tuning	✅	❌	$$$	Consistency

Thank You

Feel free to reach out:

- https://www.linkedin.com/in/dario-flores-20a426a9/
- https://github.com/dariofl24
- https://www.griddynamics.com/



Pythonistas GDL
Pythonistas Community