Beyond the Hype: What Post-Training Can (and Can't) Do for LLMs

Without Breaking Your Model, Budget, or Spirit

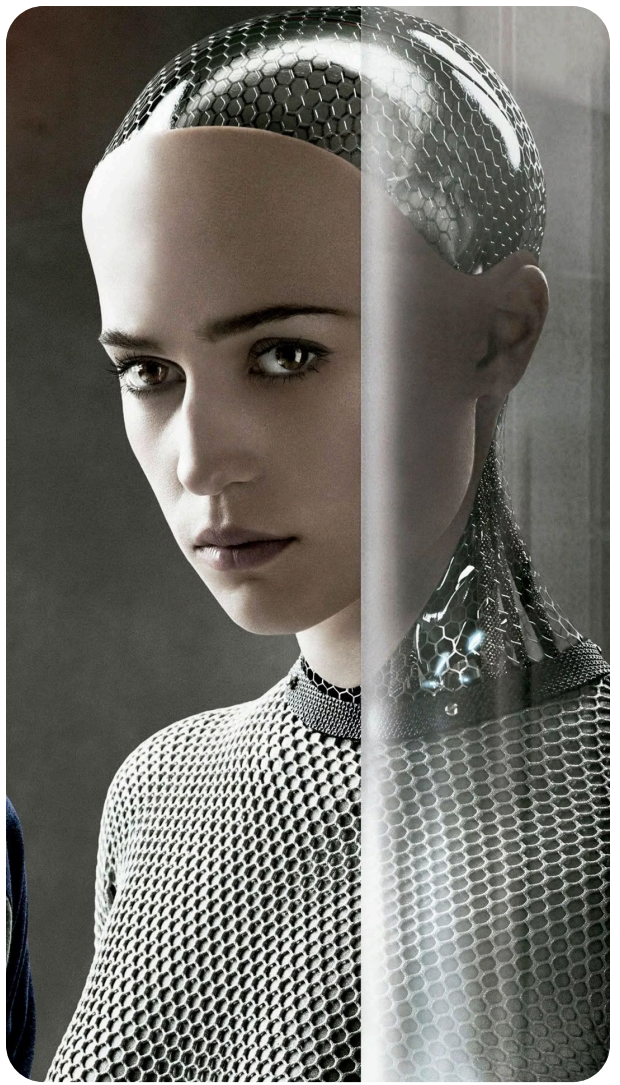
Hi, I'm Your Guide to Not Messing This Up

Welcome!

You use ChatGPT daily. You've probably asked it to write your emails, debug your code, or explain why your houseplant died. But have you ever wondered what happens under the hood? Or why sometimes it's brilliantly helpful and other times it's... confidently wrong about basic facts?

Today we're diving into the world of post-training — the art of teaching AI models new tricks without breaking them. We'll debunk myths, decode jargon, and figure out when to use RAG vs. fine-tuning vs. just writing a better prompt.

Spoiler: It's not all magic, and "just add more data" isn't always the answer.



What We're Covering Today

01

Myth-Busting Theater

Debunking the wildest misconceptions about post-training with humor and real talk

02

Decision Framework

A flowchart for humans: when to fine-tune, when to RAG, when to just... write a better prompt

03

ML Glossary for Mortals

Tokens, embeddings, and attention explained like you're talking to your funniest friend

04

How LLMs Actually Work

The 1000-foot view of what's happening inside that neural network brain

05

Post-Training Methods

Your toolkit: prompt engineering, RAG, fine-tuning, function calling, and when each one shines

06

Key Takeaways

The punchy, memorable stuff you'll actually remember tomorrow

Myth #1: Fine-Tuning = Adding Knowledge

"If I fine-tune ChatGPT on our Q4 budget, it'll know all the numbers!"

Reality check: Fine-tuning doesn't install new facts like RAM in a computer. It teaches behavior, tone, and patterns. Think of it like teaching your dog a new trick — you're not uploading "knowledge of squirrels" into its brain, you're reinforcing responses.

If you need ChatGPT to know next year's budget, use RAG to give it the actual document. Otherwise you're just training it to *sound* confident about numbers it's making up.

The metaphor: Training ChatGPT to know real-time news is like training your dog to do your taxes — adorable effort, completely wrong tool.

Myth #2: More Data = Better Model

The Assumption

"Let's just throw ALL our data at the model. More is better, right?"

The Reality

More bad data = more bad outputs. Quality beats quantity every single time.

The Truth

It's like cooking: adding more salt doesn't make food better. It makes it inedible.

Fine-tuning on messy, contradictory, or irrelevant data is like studying for a test by reading the entire internet. You'll learn some stuff, sure, but mostly you'll learn chaos. Curate your training data like you're Marie Kondo-ing your closet — does this data spark *actual value*?

Myth #3: RAG is Basically Cheating

"Using RAG means the model isn't really learning!"

Hot take: RAG isn't cheating — it's *smart problem-solving*. Retrieval-Augmented Generation gives the model access to external knowledge at runtime, like letting someone use a calculator during a math test.

Would you rather have an AI that "knows" outdated information from its training, or one that can pull up-to-date, verified sources when answering your question? RAG is giving ChatGPT the answer key *because otherwise it will confidently guess* like that one friend who never admits they don't know something.

Use RAG when: Your data changes frequently, you need citations, or the info wasn't in the training data.

Myth #4: You Can Train Away Hallucinations

Hallucinations — those moments when ChatGPT invents facts with the confidence of a game show host — are baked into how LLMs work. They're next-token prediction machines, not truth oracles. Fine-tuning won't magically fix this because hallucinations aren't a bug; they're a feature of the architecture.

Trying to eliminate hallucinations with more training is like adding extra cheese to a burnt pizza and expecting it to un-burn. Sure, the cheese is nice, but the pizza is still crispy in the wrong way.

What *actually* helps: better prompting, RAG for grounding in facts, confidence scoring, and teaching the model to say "I don't know" (yes, really). You can reduce hallucinations, but you can't train them into oblivion.

Myth #5: Models Understand Your Vague Vibes

"Just make it sound more... professional?"

What you said

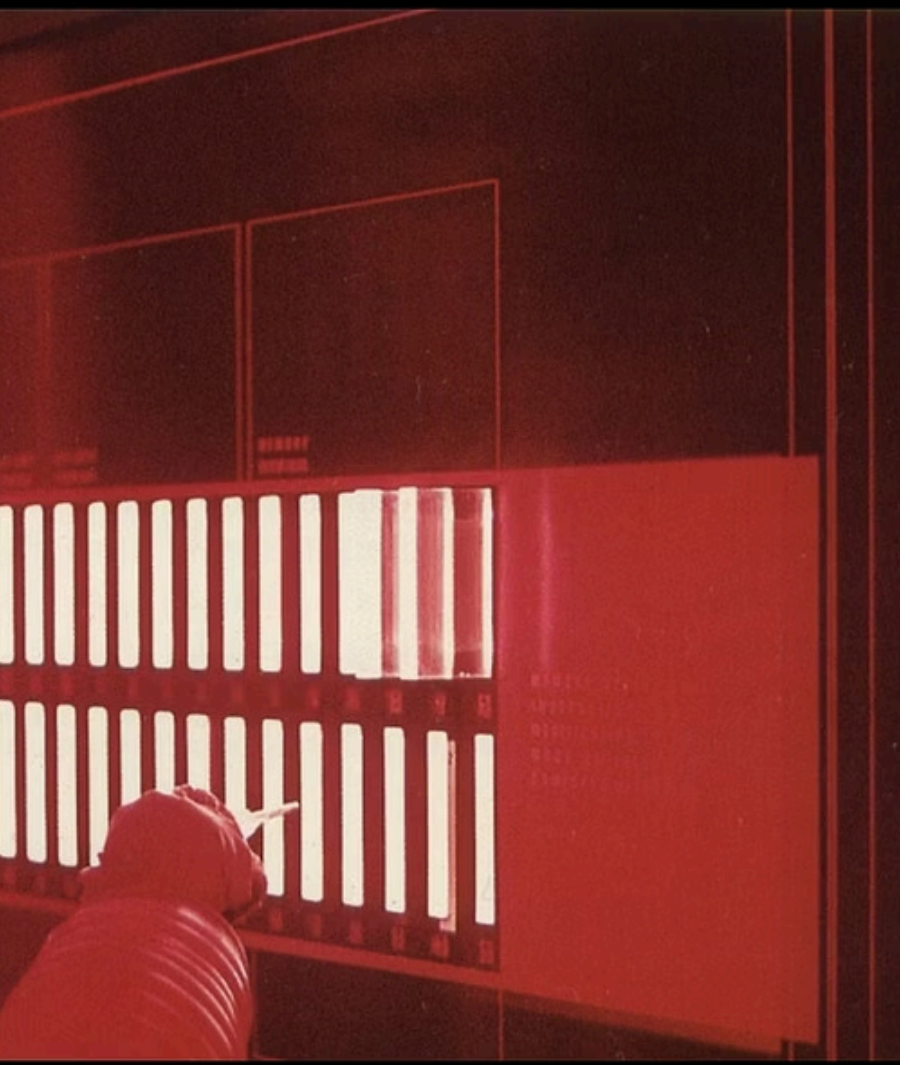
"Execute undefined_vibe.exe... ERROR 404"

What the model heard

LLMs are pattern-matching machines. They need clear, specific instructions. Saying "make it sound professional" is like telling a chef to "make it taste good" — technically a goal, but useless as guidance.

Instead of hoping fine-tuning will telepathically understand your brand voice, give examples. Show the model what "professional" means in your context with real samples. Fine-tuning can learn patterns, but it needs actual data to learn from, not vibes and wishes.

Pro tip: If your prompt is vague, your output will be vague. Garbage in, artisanal garbage out.



The Decision Tree: What Do I Actually Need?

Problem: Outdated or Missing Info

Solution: RAG — Give the model access to external, current knowledge bases

Problem: Wrong Tone or Style

Solution: Fine-tuning or System Prompts — Teach it how to talk, not what to know

Problem: Bad at Math/Logic/APIs

Solution: Function Calling — Let code handle code things, not the language model

Problem: Prompt Isn't Working

Solution: Better Prompt Engineering — Seriously, try rewriting it first before going nuclear

ML Glossary for Normal Humans

Let's decode the jargon with metaphors that actually make sense.

Model

The "brain" created after training. It's basically a mathematical pattern-matcher.

Tokens

Breaking text into crumbs so the model can digest it. "ChatGPT" becomes ["Chat", "G", "PT"]. Like breaking a samosa into pieces so your brain processes it slowly instead of choking.

Embeddings

The model's secret love language. Words get converted into numbers that capture meaning. "King" and "Queen" live close together in math space. It's how ChatGPT knows "cat" is more similar to "dog" than "chair."

Weights

The billions of tiny knobs inside the model that get adjusted during training. They determine how the model processes inputs. Think of them as the model's personality settings.

Attention

The part of the model that decides which words matter most in a sentence. Unlike your friend who ignores your texts, attention actually focuses on important stuff. It's why "not" changes the meaning of "I am happy" dramatically.

Training

Feeding the model lots of data so it can learn patterns. Like teaching a dog tricks, but with math.

Inference / Prediction

Using the trained model to answer questions or make decisions. The model's "showtime."

Labels

The answer the model is supposed to learn. The actual house price in training data.

Overfitting

When the model memorizes training data instead of learning to generalize. Like a student who memorizes past exam answers but fails new questions.

Underfitting

When the model hasn't learned enough — too simple to capture useful patterns. The opposite problem.

Parameters

The "knobs" inside the model that get tuned during training. (Similar to weights but more general)

Bias

A systematic mistake. Model favors one type of outcome unfairly.

Evaluation Metrics

Scores to measure how good the model is. Examples: accuracy, precision, recall, F1.

How LLMs Actually Work: The 1000-Foot View

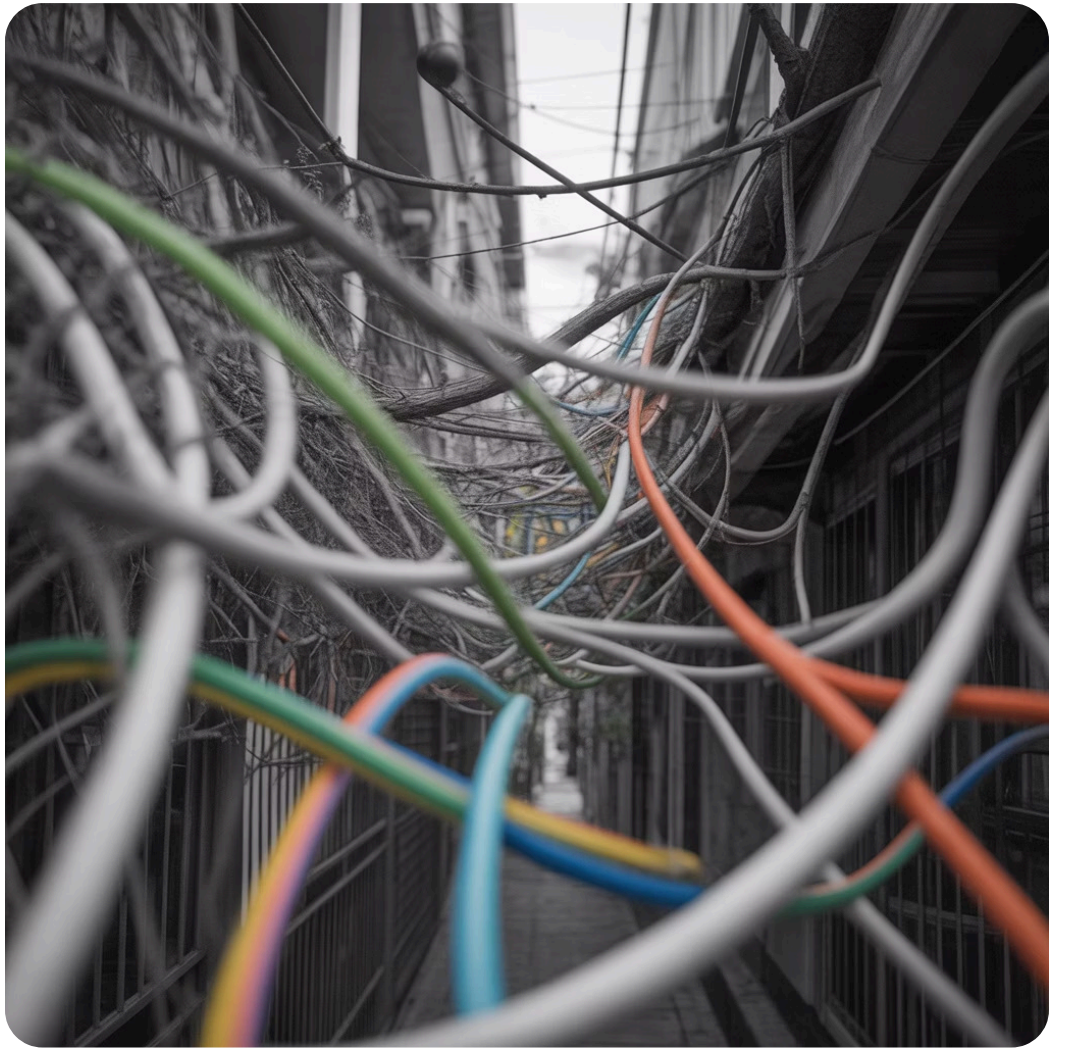
Tokenization: Breaking It Down

First, your text gets chopped into tokens. Every input becomes a sequence of numbers. "Hello world" might become [15496, 995]. The model only speaks numbers internally, like a very sophisticated calculator that learned to talk.

Next-Token Prediction

At its core, an LLM is playing a game of "finish the sentence" trillions of times. Given "The cat sat on the...", it predicts "mat" has a high probability. It's not understanding — it's pattern matching at industrial scale.

Key insight: ChatGPT isn't your soulmate finishing your sentences. It just read billions of words and learned the vibes.



Training: The Big Picture

Pre-training

Reading the entire internet (well, a big chunk). The model learns language, facts, and patterns. Costs millions of dollars and thousands of GPUs. This is where it gets its "general knowledge."

1

2

Fine-Tuning

Teaching specific behaviors on smaller datasets. Like sending the model to finishing school to learn manners, tone, or specialized tasks. Way cheaper than pre-training.

3

Alignment/RLHF

Training the model to be helpful, harmless, and honest using human feedback. This is why ChatGPT says "I'm just an AI" instead of going full HAL 9000 on you.

Training isn't memorization — it's more like reading 2 million books and remembering just the vibes, the patterns, the general shape of human knowledge. The model can't quote every book, but it learned how language works.

Transformers: The Architecture That Changed Everything

Transformers are the secret sauce behind GPT, Claude, and basically every modern LLM. The breakthrough? **Attention mechanisms** that let the model focus on relevant parts of the input, no matter how long it is.

Before Transformers, models processed text sequentially like reading a book word-by-word with a finger on the page. Transformers can look at the whole sentence at once and decide which words are BFFs and which ones are just background noise.

Why this matters: Attention is why ChatGPT can handle long contexts and understand that "it" in sentence 47 refers to "the database" mentioned way back in sentence 3. It's the model's version of stalking the most important words in a sentence.

Application Method #1: Prompt Engineering

This is NOT post-training — it's how you USE an already-trained model. But it's your first line of defense before considering actual post-training.

What It Is

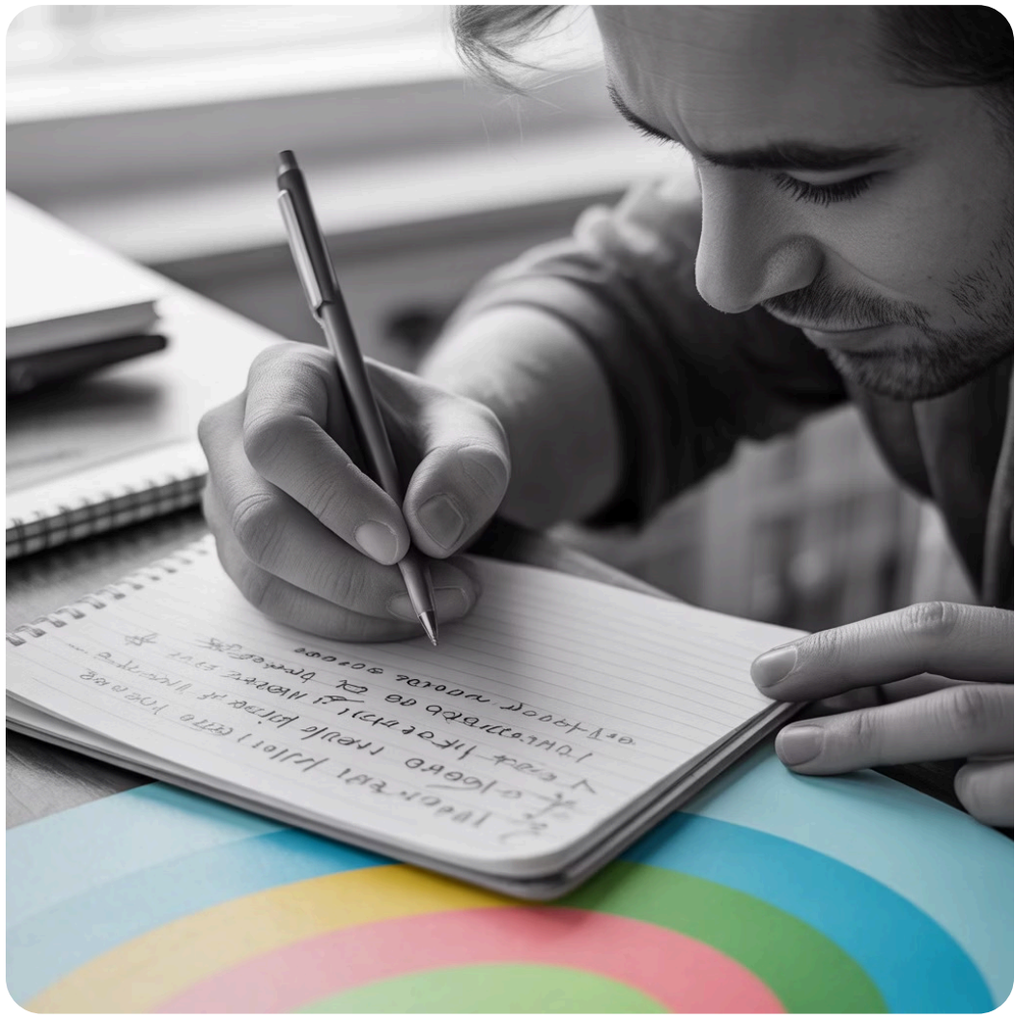
The art of writing better instructions. It's free, fast, and shockingly effective. Before you reach for fine-tuning or RAG, try rewriting your prompt with more context, examples, and structure.

When It Works

- You need quick iteration
- The model already has the knowledge
- You just need to guide its behavior
- Budget is tight (it's free!)

When It Fails

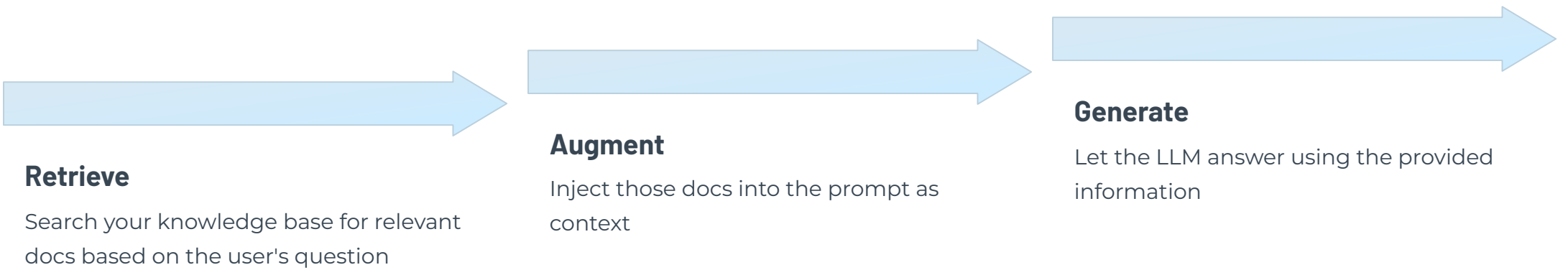
When the model doesn't have the information, no amount of prompt wizardry will make it materialize. Also fails when you need consistency across thousands of requests.



The metaphor: Like bribing ChatGPT with clarity — the more specific details you give, the less chaos you get back.

Application Method #2: RAG (Retrieval-Augmented Generation)

RAG is NOT post-training — it's an inference-time technique. You're not changing the model's weights, you're giving it better context.



Retrieve

Search your knowledge base for relevant docs based on the user's question

Augment

Inject those docs into the prompt as context

Generate

Let the LLM answer using the provided information

When to use RAG: Your data changes frequently (company docs, news, real-time info), you need source citations, or the information wasn't in the model's training data. It's like giving the model the answer key because otherwise it will confidently guess like that one friend who never admits they're wrong.

When it fails: When your retrieval system returns garbage, or when the answer requires reasoning beyond just looking up facts. RAG can't teach the model new behaviors or writing styles.

Post-Training Method #1: Fine-Tuning (SFT)

THIS is actual post-training — you're modifying the model's weights with your data.

The Custom Tailoring Approach

Fine-tuning adjusts the model's weights on your specific dataset. It's useful when you want the model to talk EXACTLY like your brand, follow specific formats consistently, or handle specialized tasks.

When it works: You need consistent tone/style across thousands of outputs, you have quality training data, and you're teaching behavior rather than facts. Like giving ChatGPT etiquette classes so it sounds corporate, casual, or like a 1920s detective.

When it fails: You're trying to teach new facts (use RAG instead), your training data is tiny or low-quality, or you just needed better prompting. Fine-tuning is overkill if a system prompt would do the job.

Pro tip: Start with prompt engineering, graduate to fine-tuning only when you've proven the need at scale.

Application Method #3: Function Calling & Tools

Like RAG and prompting, this is NOT post-training. It's a way to extend what a trained model can do at inference time.

What It Is

Let the model call external tools, APIs, or code instead of trying to do everything itself. ChatGPT figures out WHEN to call a function, and your code does the actual work.

Perfect For

Math, database queries, live data fetching, or anything requiring precision. ChatGPT doing math raw is like asking a poet to balance your bank account.

The Power Move

Combine LLM reasoning with code execution. The model handles language, your functions handle logic. Best of both worlds.

When it fails: When the model misidentifies which function to call, or when your API is unreliable. Also, function calling adds latency, so it's not ideal for speed-critical applications.

Choosing Your Weapon: Quick Decision Guide

Need a quick fix, no budget	Prompt Engineering
Need citations & fresh data	RAG
Want consistent brand voice/format	Fine-Tuning
Need math, APIs, or live calculations	Function Calling
Frequently changing knowledge base	RAG

Match your situation to the right method. Start simple (prompts), add retrieval when you need grounding, fine-tune for style consistency, and use function calling when you need precision or external data.

Why LLMs Hallucinate (And What to Do About It)

The Architecture

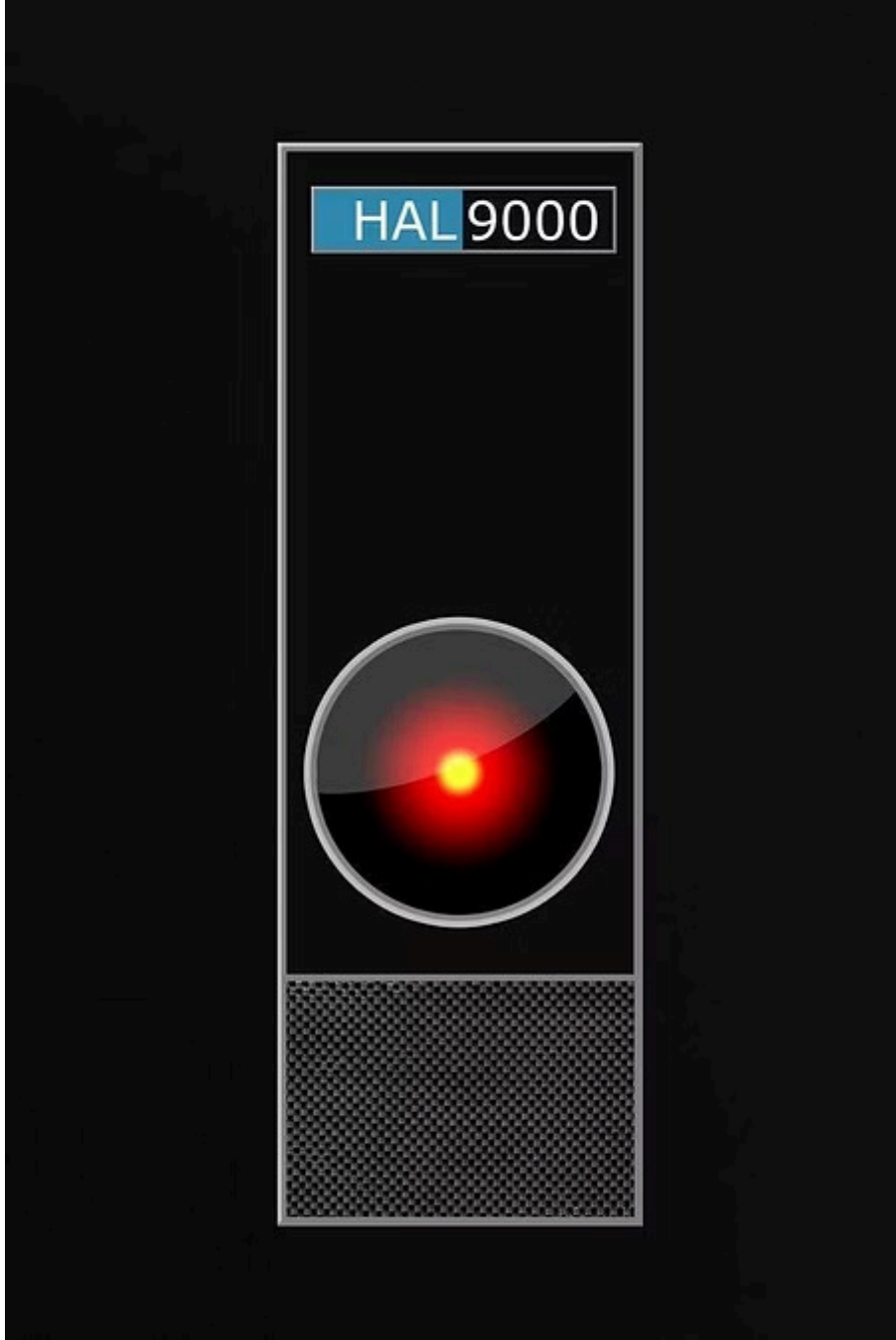
LLMs predict the next most likely token. They don't have a "truth database" — they're sophisticated autocomplete on steroids. Sometimes the most likely next word is confidently wrong.

Training Gaps

The model never saw certain information, so it interpolates based on patterns. Like a student who didn't study for a test and is winging the essay question with "vibes."

Mitigation Strategies

Use RAG to ground outputs in real documents. Add confidence scoring. Teach the model to say "I don't know." Use function calling for facts that need precision. Prompt for citations.



Key Takeaways: What to Remember Tomorrow

Post-Training ≠ Magic

It's a powerful toolkit, but use the right tool for the job. More data isn't always better. Fine-tuning won't teach new facts.

Start Simple, Scale Smart

Try prompt engineering first. Graduate to RAG for knowledge, fine-tuning for behavior, function calling for precision.

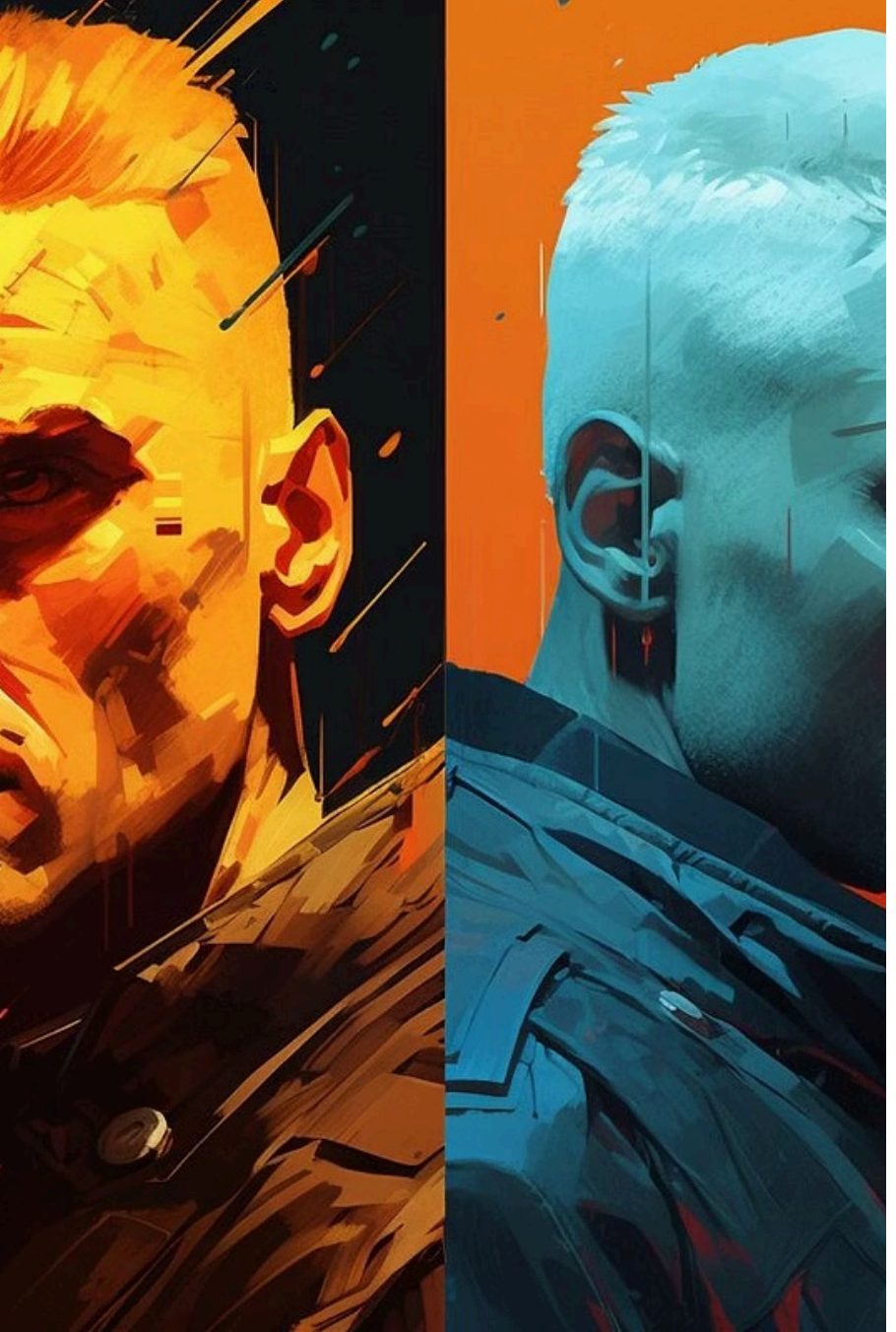
Hallucinations Are Features

Not bugs. You can reduce them with RAG, grounding, and smart prompting, but you can't eliminate them entirely.

Quality > Quantity

In training data, prompts, and everything else. Curate ruthlessly. Test relentlessly. Iterate constantly.

Final wisdom: Post-training is powerful, but it's not a magic wand. Use it when it makes sense — not when memes tell you to. Now go forth and build something awesome! 🚀



Beyond Base Models: Why Post-Training Exists

The Reality Check: Not all models need post-training — start with "WHY".

Open models are increasingly capable out of the box

Post-training is for specialization, better behavior & controllability

It's sculpting, not reinventing the wheel

Think of it as the final 10% that makes everything work smoothly

Using a base model as-is is like buying 90% assembled IKEA furniture. Post-training is the other 10%... the screws that keep the table from wobbling when life gets tough.

What Post-Training SHOULD Achieve

Instruction-following mastery

The model learns to follow complex, multi-step instructions reliably

Role switching & tone control

Seamlessly shift from Shakespeare to corporate email to casual chat

Better reasoning / math / multilingual support

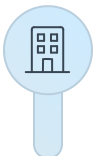
Enhanced capabilities in specialized domains

Structure & task consistency

Predictable, reliable outputs every time

Want your model to switch from Shakespeare mode → corporate email mode → Hinglish meme mode? That's post-training magic.

The Post-Training Recipe (Masterchef LLM Edition)



Step 1: Start with SFT

Reliable, cheap, predictable foundation

Step 2: Add Preference Optimization or RL

Behavior shaping and alignment

Step 3: Season with Great Data

Curated > massive. Quality beats quantity

Step 4: Taste-test with Evals

Test after each change to ensure improvement

Fun Analogy: "SFT = marinating the chicken. Preference training = adding perfect spices. Evals = the chef tasting before serving."

Step 1: Evals Before Everything Else

Why Evals Matter:
Define "good" behavior BEFORE training. You can't improve what you can't measure.

Use layered eval suites
(chat, reasoning, code, long-context, multilingual)

Keep evals updated
— expectations evolve fast

Test on real-world scenarios
not just benchmarks

Track regressions as carefully
as improvements

"Training without evals is like renovating your house with a blindfold — you'll only know it's wrong when you bump into a wall."

Choosing the Right Base Model

Size Matters (But Not How You Think)

Pick size intentionally

Bigger isn't always better. Match model size to your use case and infrastructure

Dense models easiest to fine-tune

Standard architectures have better tooling and community support

Proven track records

Choose architectures with successful post-training examples

Consider the full picture

Latency, deployment cost, inference speed, and memory requirements

"Buying a 70-inch TV for a 7-foot room is chaos. Same with picking a massive model for tiny use-cases."

Common Post-Training Mistakes (AKA the Hall of Shame)

Learn from Others' Pain

Overfitting narrow domains

Model forgets everything else. Your legal assistant can't chat anymore.

Bad evaluation design

No idea if anything improved. Flying blind.

Wrong data mixture

Great at code but terrible at chatting. Unbalanced diet.

Changing too many variables

Chaos mode. Can't tell what worked and what broke.

"Tuning too many things at once is like mixing every soda flavor at a restaurant because 'YOLO'. It never ends well."

When NOT to Post-Train (Yes, Sometimes Don't Do It)

Sometimes the Answer is "Don't"

Simple prompting solves your need

Don't overcomplicate. Start with the easiest solution.

Grand knowledge updates

Use RAG, not training. Training isn't for teaching new facts.

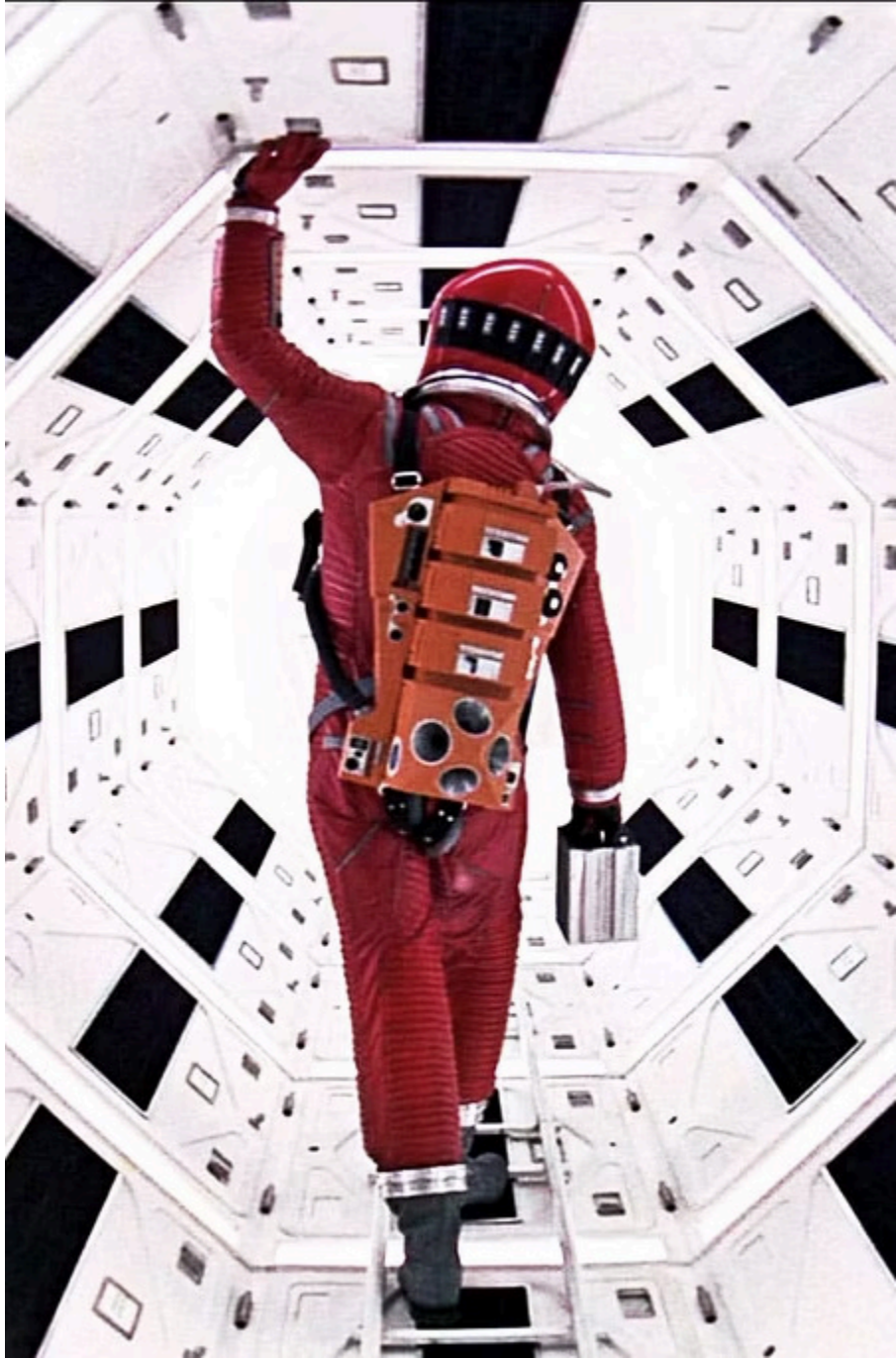
Bad or tiny data

Post-training might worsen the model. Garbage in, garbage out.

Use-case doesn't require specialization

The base model already does what you need.

"You don't need a custom sports car if all you're doing is grocery runs."



The Post-Training Cheat Sheet

Your Quick Reference Guide

Start with WHY - Have a clear reason before you begin

Evaluate capabilities → THEN train - Know your baseline first

Use SFT → PO → RL - Follow the proven recipe

Curate clean, balanced data - Quality over quantity, always

Run evals after each step - Measure progress continuously

When done right → Reliable, aligned assistant that does what you need

"Post-training isn't magic — it's cooking: pick good ingredients, taste often, and don't burn the kitchen."

Live Demo Ideas & Audience Participation

Make It Memorable

Instruction-following

"Explain quantum computing to a 5-year-old who likes dinosaurs & hates broccoli"

RAG vs No-RAG

"What's our refund policy?" (with and without knowledge base)

Tone-shift mastery

"Formal → Hinglish → Bollywood villain mode"

Tool-calling for precision

Live loan EMI calculation with real numbers

Audience Challenges:

- "Guess what the base model will do" game
- "Fix this prompt" challenge
- "Choose your training method" quiz (Prompting / RAG / SFT / LoRA / Tools)

Thank You! 🚀

Remember: You don't need bigger models — you need smarter training.

- Post-training is powerful, but it's not a magic wand
- Use it when it makes sense — not when memes tell you to
- Start simple, measure everything, iterate constantly
- Now go forth and build something awesome!

"LLM before vs after post-training = the ultimate glow-up"

Questions? Let's chat!

(Preferably not about quantum physics unless we have broccoli)