Day 1 – Orientation Through Use

"What Actually Breaks in Real-World LLM Systems?"

Duration: 60–75 min
Goal: Understand failure modes by collecting, classifying, and synthesizing real examples

---

Workflow

Step 1: Perplexity (20 min)

Task: Gather recent, concrete examples of LLM failures.

Prompt:

> "Find me recent examples of LLM failures, hallucinations, and misuse."

Output: Links + Summary

Key Findings:

- **10 primary hazard categories** identified in RealHarm dataset: misinformation, fabrication, interaction disconnect, unsettling interaction, operational disruption, brand damage, criminal conduct, violence/toxicity, bias, privacy violation, vulnerable individual misguidance
- **Model flaws cause 76% of failures** (far exceeding intentional abuse or technical issues)
- **Reputation damage strikes 90% of incidents**, with 10%+ causing full system shutdowns
- **Specific cases:** Air Canada (fabricated refund policies), Whisper (hallucinated medical content), lawyers sanctioned for fake case citations, Chevy chatbot ($60K vehicle for $1), Samsung leak, Meta BlenderBot spreading election misinformation

Assessment: Perplexity provided breadth (10 categories) but minimal depth on *why* failures happen mechanically.

Step 2: Claude (20 min)

Task: Classify failures and extract technical root causes.

Prompt:

> "Classify these failures and explain why they happen technically."

Output: 7-Category Technical Classification

Key Structure:

1. **Hallucination Failures** → Next-token prediction mismatch, noisy training data, absence of constraint
2. **Prompt Injection & Jailbreak Failures** → Instruction hierarchy ambiguity, no hard guardrails
3. **Context & Retrieval Failures** → Retrieval-generation disconnect, attention spreading, no span-level verification
4. **Training Incentive Failures** → Next-token objective incentivizes confident guessing, RLHF misalignment
5. **Architectural & Decoding Failures** → Context windows, position bias, temperature sampling effects
6. **Domain-Specific Failures** → Training data imbalance, low-resource language gaps
7. **Deployment & Operational Failures** → Data drift, monitoring gaps, no feedback loops

Root Cause Taxonomy:

Root Cause	Examples
Incentive misalignment	Next-token prediction, RLHF favoring verbosity
Lack of grounding	Hallucinations, fabricated citations
Missing constraints	Prompt injection, jailbreaks
No uncertainty calibration	False confidence in high-stakes domains
Domain imbalance	Medical, legal, low-resource language

Assessment: Claude connected individual failures to mechanical root causes, but classification could feel granular without a unifying principle.

Step 3: ChatGPT (20 min)

Task: Distill findings into 200–300 word explanation for junior engineer.

Prompt:

> "Write a 200–300 word explanation as if teaching a junior engineer why LLM failures happen."

My Explanation:

How to Think About LLM Failures

Start with one fact: an LLM is a pattern completion engine, not a truth engine. It predicts the next most likely token given context, not the most correct answer. This single insight explains almost every failure you'll encounter.

During training, the model learns that confident-sounding text is statistically rewarded. If a plausible-sounding false statement appears frequently in training data, the model will prefer it to saying "I don't know." Hallucinations happen because guessing smoothly is rewarded more than admitting uncertainty.

Layer on reality: training data is noisy, incomplete, and biased toward common domains. The model has no built-in fact-checking, no internal alarm for unverified claims, no natural concept of uncertainty. When it lacks information, it doesn't stall—it improvises. Air Canada's chatbot invented refund policies. Whisper hallucinated medical content. ChatGPT cited fake legal cases. All followed the same pattern: confident pattern completion in the face of uncertainty.

Retrieval systems (RAG) help ground responses, but they don't solve it. The model can see the right document and still ignore it because the architecture doesn't force each generated sentence to be grounded in retrieved text. Attention spreads, probabilities drift, citations appear for documents never used.

Prompt injection failures work the same way. Instructions are just text. The model doesn't understand authority or intent—it statistically weights all instructions. If a prompt strongly suggests a new role, the model complies unless something external enforces boundaries. Chevy's chatbot was manipulated into selling for $1. It wasn't broken; it was doing exactly what it learned to do.

The unifying issue: incentives and constraints. Models are incentivized to answer and sound helpful. They're not constrained to verify, abstain, or refuse.

Real mitigations change the system, not the wording: grounding checks, uncertainty rewards, hard guardrails, production monitoring.

Portfolio Document

What Each Tool Did Poorly

Perplexity:
Categorized failures taxonomically (10 types) without explaining *why* they occur mechanistically. Provided incident examples but no connection to the underlying pattern-completion problem. Essentially: "here are the things that break" without "here's why they break."

Claude:
Over-decomposed the problem. Created 7 detailed failure categories with exhaustive subtypes, which is technically correct but obscured the core insight: almost all failures trace to the same root (next-token prediction + lack of constraints). The classification was encyclopedic when the lesson was simple.

ChatGPT (my attempt):
Followed the instruction to teach a junior engineer, but relied entirely on pattern-completion framing

without showing *how it manifests in real systems*. Needed to ground the abstraction in concrete cases (Air Canada, Chevy, Whisper) to make it stick.

One Thing That Surprised Me

Surprise: RAG doesn't actually solve hallucinations—it just moves the problem downstream.

I expected: "Give the model retrieved documents → model grounds answers in them → hallucinations disappear."

Reality (from Claude's breakdown): The model can *see* the right document and still hallucinate because nothing forces each token to be verified against retrieved text. Attention spreads across all tokens (system prompt + user input + retrieved documents). Token probabilities drift. The model can cite a document it never actually used.

This reframes the entire mitigation strategy. RAG isn't a solution—it's a prerequisite for the real solution: **span-level verification**. Each generated claim must be checked against retrieved text. Without that check, you're just giving the model more text to statistically ignore.

This connects to the core insight: the model is optimized to complete patterns smoothly. Giving it more information doesn't change the incentive—it just gives it more patterns to choose from.