1. How to increase accuracy, reliability & make answers verifiable in LLM?

To move an LLM from a "creative writer" to a "reliable expert," you need to constrain its generation and force it to cite evidence.

- **Grounding (RAG):** Instead of relying on the model's internal memory (which hallucinates), provide a "Source of Truth" (documents, database) in the prompt and force the model to answer *only* using that context.
- **Citations:** Instruct the model to cite the specific document or chunk ID for every claim.
    - *Prompt Example:* "For every sentence you generate, append the source ID like [Source 1]."
- **Chain of Thought (CoT):** accuracy improves when the model is forced to explain its reasoning step-by-step before giving the final answer. This reduces logic errors.
- **Self-Consistency / Voting:** Generate 5 answers for the same prompt (with high temperature) and pick the most common answer. This is statistically more reliable than a single generation.
- **Verifiability:** Use "Evan" (Evaluation) frameworks like **Ragas** or **DeepEval**. You can treat the LLM's output as a claim and use a second "Judge LLM" to verify if the claim is supported by the retrieved context.

---

2. How does RAG work?

Retrieval-Augmented Generation (RAG) is a technique that connects a generative model (like GPT-4) to an external database. It works in three phases:

1. **Retrieval:** When a user asks a question, the system searches a vector database for relevant information (chunks of text) that matches the user's query intent.
2. **Augmentation:** The retrieved chunks are pasted into the prompt as "Context."
3. **Generation:** The LLM receives the prompt containing both the User Query and the Retrieved Context and generates an answer based *only* on that context.

---

3. What are some benefits of using the RAG system?

- **Up-to-Date Knowledge:** LLMs have a "training cutoff" (e.g., knowledge stops at 2023). RAG allows the model to answer questions about data created *today* (e.g., stock prices, new emails) without retraining.
- **Reduced Hallucination:** By forcing the model to rely on retrieved context, you significantly lower the chance of it making up facts.
- **Source Citation:** Since the system knows exactly which documents were retrieved, it can show the user the original source (e.g., "See Page 5 of the PDF").
- **Data Security:** You can keep your proprietary data in a secure database and strictly control access (ACLs) during the retrieval step, ensuring the LLM respects user permissions.

---

4. When should I use Fine-tuning instead of RAG?

This is the classic "Knowledge vs. Behavior" trade-off.

Feature	RAG (Retrieval)	Fine-Tuning (Training)
Goal	**New Knowledge:** The model needs to know facts it wasn't trained on (e.g., your company's Q3 report).	**New Behavior/Style:** The model needs to talk in a specific tone, code in a specific internal language, or follow a strict JSON format.
Data Dynamicness	**High:** Data changes daily. You just update the database.	**Low:** Data is static. Retraining is expensive and slow.
Hallucination	**Low:** Grounded in retrieval.	**Medium:** The model can still hallucinate facts even after fine-tuning.
Cost	**Low:** Only inference and storage costs.	**High:** Requires GPUs for training and hosting custom models.

Verdict:

- Use **RAG** for factual accuracy and accessing proprietary documents.
- Use **Fine-Tuning** if you need the model to sound like a specific persona (e.g., a medical assistant) or learn a new language/code syntax.
- *Pro Tip:* Often, the best architecture is **Hybrid** (Fine-tune a model to be a good RAG reasoner, then use RAG for the facts).

---

5. What are the architecture patterns for customizing LLM with proprietary data?

There are three main patterns for injecting private data into an LLM application:

Pattern A: Context Injection (RAG)

- **Mechanism:** Search a database $\rightarrow$ Stuff results into the Prompt $\rightarrow$ Ask LLM.
- **Best For:** QA bots, Search engines, applications requiring citations.

- **Pros:** Cheapest, easiest to update, highly verifiable.

Pattern B: Small Language Model (SLM) Fine-Tuning

- **Mechanism:** Take a small open-source model (e.g., Llama-3-8B) and train it on your specific domain data (e.g., medical records).
- **Best For:** Privacy-heavy on-premise deployments, or when specific jargon/vocabulary is critical.
- **Pros:** Data never leaves your server, highly specialized performance.

Pattern C: Hybrid (RAG + Fine-Tuning)

- **Mechanism:** Fine-tune a model specifically to understand your domain's *structure* (e.g., how to read your complex financial tables), and then use RAG to feed it the actual *numbers*.
- **Best For:** Complex enterprise use cases where off-the-shelf models fail to understand the nuance of the documents.