Advanced NLP -

Session 6: LLM Engineering

Prof. Dr. Richard Sieg
TH Köln IWS - WS 25/26

Lecture	Date	Topic
1	03.12.2025	Introduction & NLP Recap
2	04.12.2025	RNNs and LSTMs
3	10.12.2025	Attentions & Transformers
4	11.12.2025	Transformer Based Models
5	17.12.2025	Hackathon / Check-In
6	18.12.2025	LLM Architecture
7	07.01.2026	LLM Engineering
8	08.01.2026	Hackathon / Check-In
9	14.01.2026	LLM Shortcomings
10	15.01.2026	Final Presentations

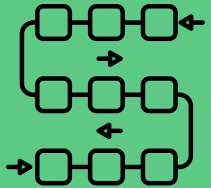
Agenda

01. Tutorial: LLM Engineering

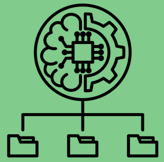
02. RAG

03. Evaluating LLM Applications

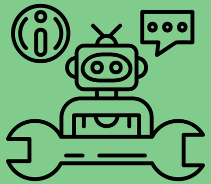
The 3 Ingredients of LLMs



Process long sequences and context



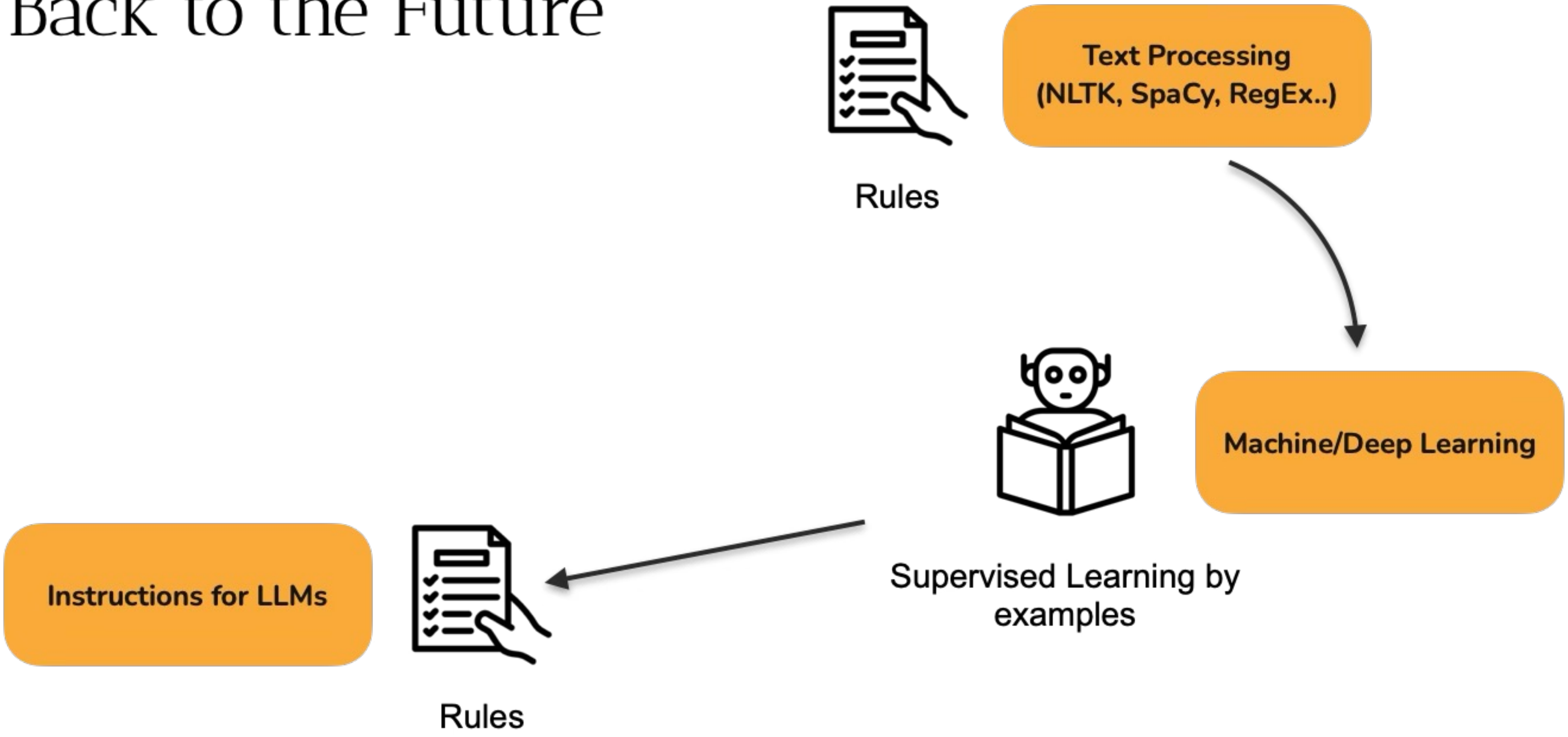
Efficient training on huge datasets



Follow (human) instructions

Effectively

Back to the Future

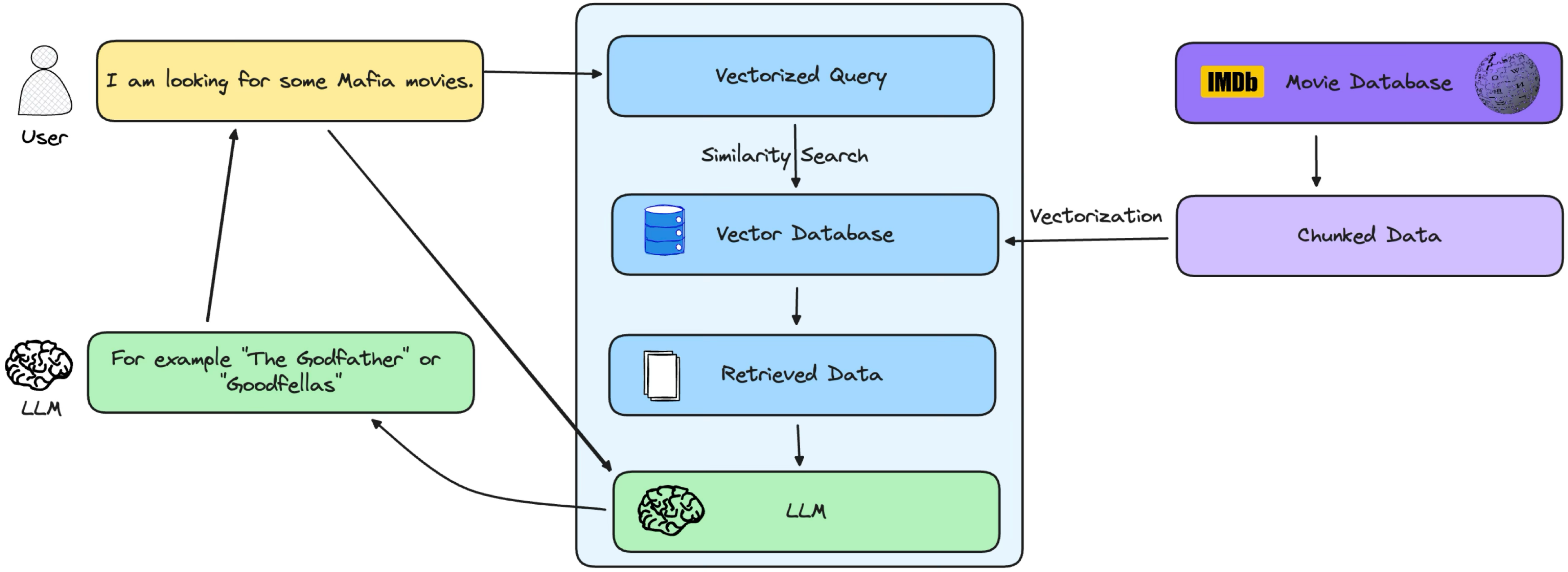


Tutorial

How can we give an LLM access to data?

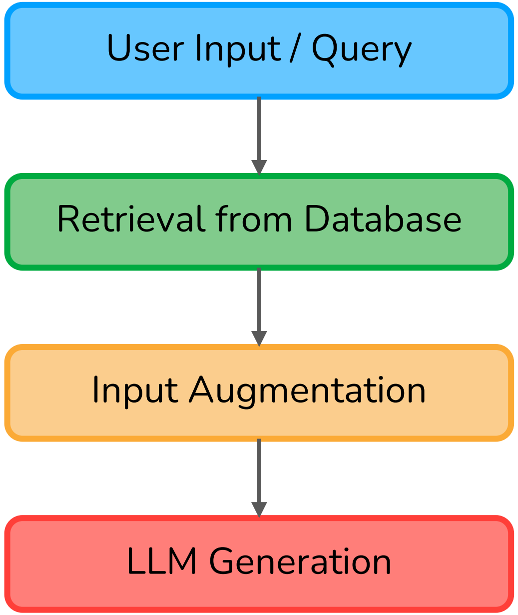
02.

Retrieval Augmented Generation (RAG)


User
I am looking for some Mafia movies.
Vectorized Query
Similarity Search
Vector Database
Retrieved Data
LLM
For example "The Godfather" or "Goodfellas"
LLM
IMDb
Movie Database
Chunked Data
Vectorization

RAG: Main Idea

- Based on the user input, relevant documents are ***retrieved*** from a database
- Those documents are given to the LLM as additional context (***augmentation***)
- The LLM then ***generates*** the output based on the original input and retrieved documents



⚠️ There are some frameworks that unify all of these steps but in the end I always ended up building it myself

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks [Lewis et. al., 2021]

Step 1: Indexing

- Depending on the size of the documents and the context window of the LLM, we might need to chunk the documents first
  - Different chunking strategies: Number of characters, overlaps, headings etc
- Compute vector embeddings for each chunk using a “small” model
  - Popular choices: (Sentence)BERT, models by LLM providers
- Store those embeddings along with the original text and metadata in a vector database
  - Popular choices: qdrant, Weaviate, Chroma, Pinecone, Faiss, Cohere

Step 2: Retrieval

- Use the LLM input as a query for the vector database
- Embed the query using the same model as for embedding the documents
- Using ***Similarity Search***, retrieve the k most similar documents from the database, e.g. using ***cosine similarity***

$$sim(query, document) = \frac{q \cdot d}{\|\mathrm{q}\| \cdot \|\mathrm{d}\|}$$

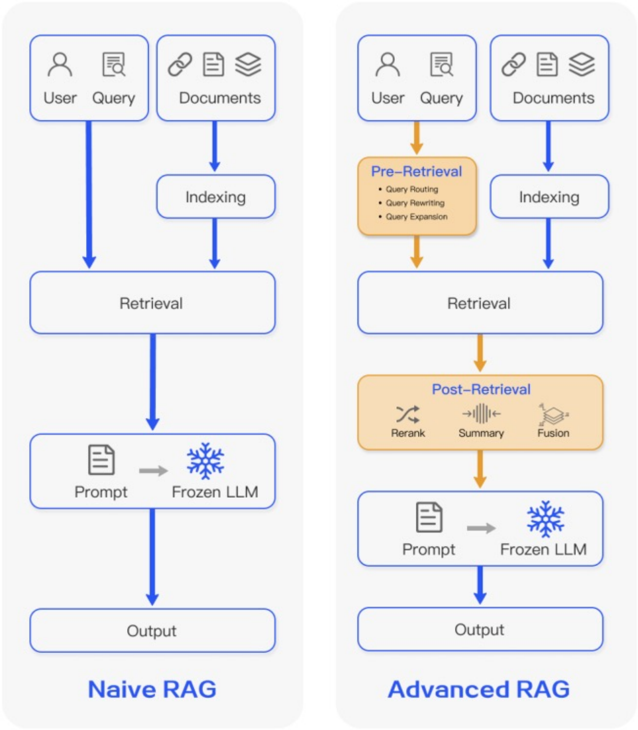
- This can be combined with standard retrieval techniques like keyword search

Step 3: Generation

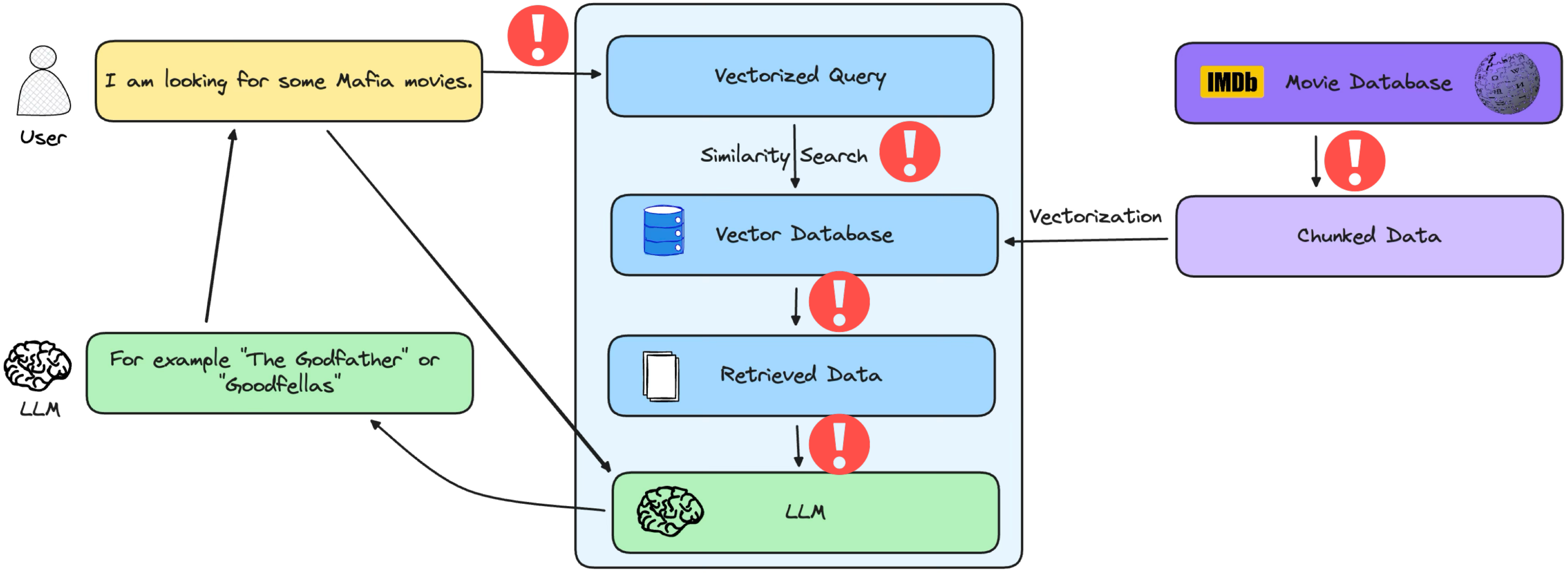
- The original input is augmented with the retrieved chunks and given to the LLM
- The retrieved chunks are indicated in the augmented prompt:
  `### CONTEXT` or **`<context> … </context>`** (see tutorial)
- It is often a good idea to note in the system prompt that the LLM will receive context from a database

Extended RAG Techniques

- **Indexing**
  - Different chunking strategies, use different embeddings (full text, summary etc), metadata, hierarchical index
- **Retrieval**
  - Rewrite query, query routing to different sources
- **Generation**
  - Rerank retrieved sources, build summaries, context selection



RAG Challenges



RAG Challenges

- Key questions
  - *When and what should be retrieved? How is the context used?*
- Chunking strategy tradeoffs (size, overlap etc)
- “Internal Knowledge” of the LLM vs information in the database
- Robustness with incorrect information in the documents
- Quality and relevance of the context, context too noisy
- What is the best embedding model for the use case?
- Ever-growing context window of LLMs (“RAG is dead 🪦”)
- ”Lost in the middle” effect – information from the middle of a context is less likely used by the LLM

Prompt Engineering or Fine-Tuning?

Prompt Engineering

Tell the model what to do at runtime, every time
"You are an expert in..."

Benefits

- Adaptable – for many inputs
- Instant – no training
- Flexible – change anytime
- Experimenting is cheap

+ RAG
Give access to current knowledge

👉 **You should start here**

- Inefficient
- Worse performance than fine-tuning
- Sensitivity to wording
- Lack of clarity

Fine-Tuning

Modify model weights through training examples
Permanent behavior change

Only when

- Fixed & specific task
- A smaller model is needed:
- Lower latency
- Smaller costs

👉 **Only in rare cases**

Outlook: Other Important Techniques

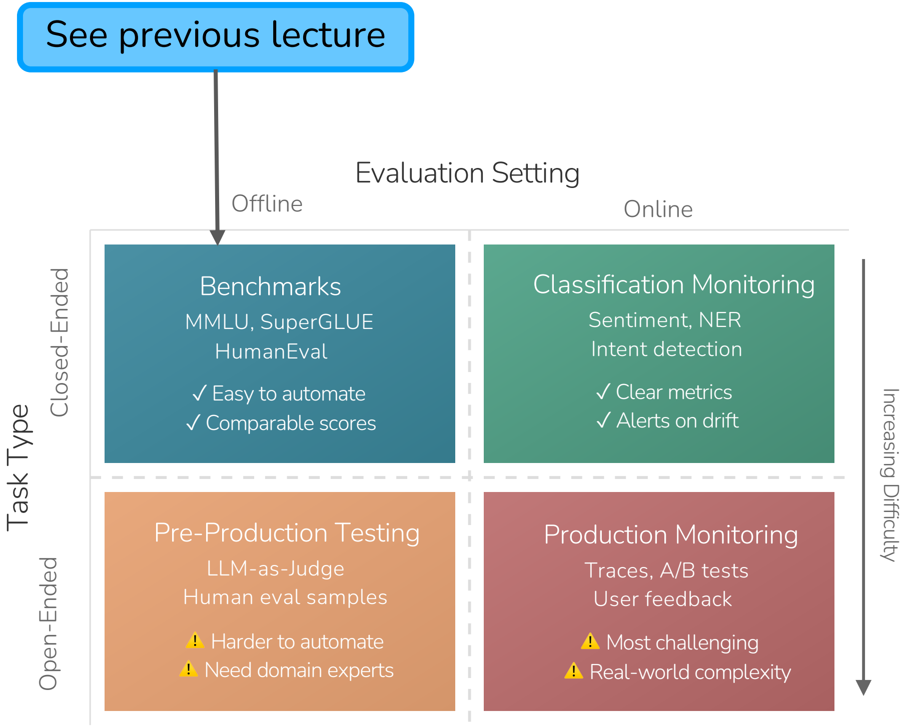
- ***Tool Use***
    - LLMs given access to a functions
- ***Agents***
    - “LLM that runs tool in a loop to achieve a goal” – Simon Willison
- ***Context Engineering***
    - Giving agents the correct context
- ***Model Context Protocol (MCP)***
- ***DSPy***
    - Automatic prompt optimization

03.

Evaluating LLM Applications

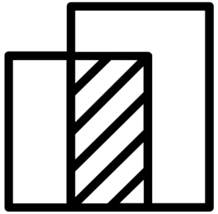
The Evaluation Challenge

- Two major evaluation paradigms
- ***Discriminative***: Closed-ended evaluations
  - Classification, QA, extraction etc
  - Limited correct answers
  - Automatic evaluation possible
- ***Generative***: Open-ended evaluations
  - Generation, conversations
  - Many valid answers
  - Evaluation is inherently difficult



High benchmark scores don't guarantee real-world performance. Models can exploit spurious correlations (e.g., negation words → contradiction) without truly understanding the task.

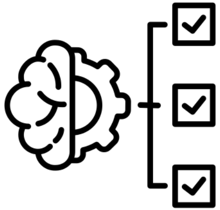
Evaluating Text Generation



Content Overlap Metrics
(BLEU, ROUGE etc)

- Measure n-gram overlap with reference text
- Fast, cheap, and widely used
- Problem: no semantic understanding

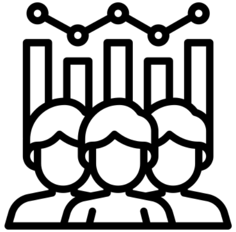
Needs Reference Text



Model Based Metrics
(BERTScore, BLEURT etc)

- Use embeddings for semantic similarity
- Better correlation with human judgement

Needs Reference Text

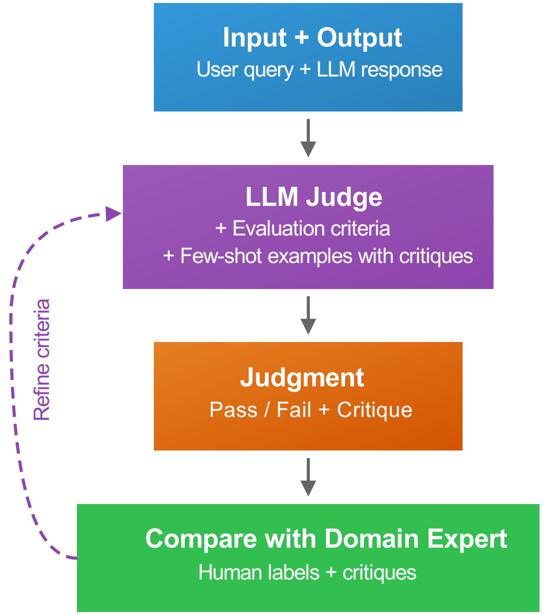


Human Evaluation

- Gold standard
- Expensive and slow

LLM as a Judge

- Issue: Human annotation does not scale
- ***LLM as a Judge***
  - LLM receives input, output, and evaluation criteria
  - LLM produces judgement and explanation (critique)



Common Pitfalls

- Position bias (prefers first answer)
- Length bias (prefers longer answer)
- Self-preference (LLM prefers own output)

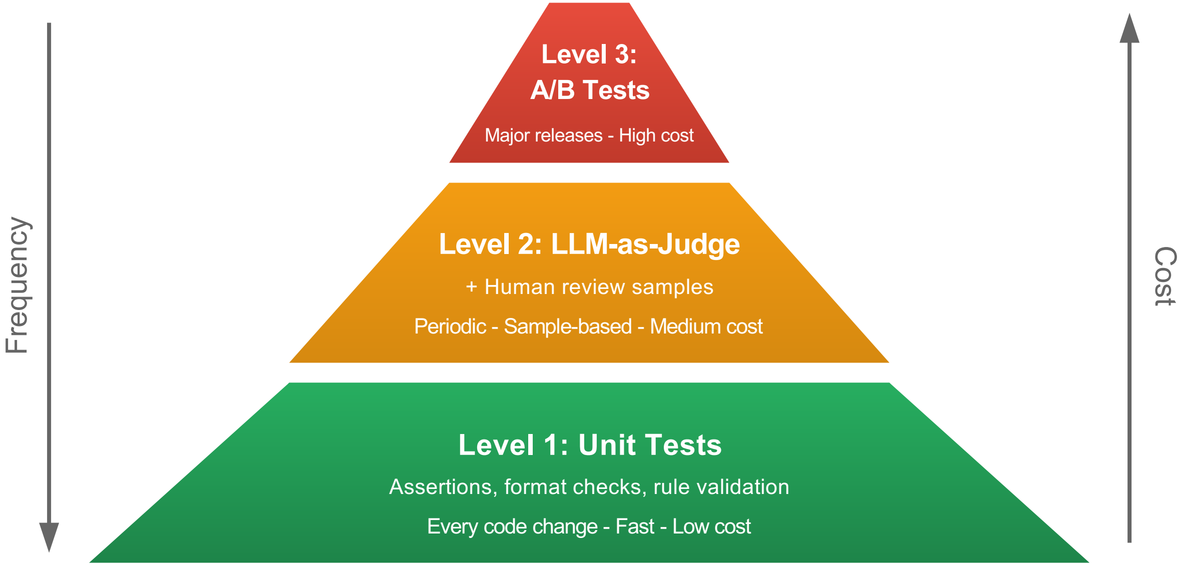
Best Practices

- Binary pass/fail decision
- Include critique
- Use task-specific criteria

Production – Traces & Monitoring

- Logging & Traces
  - Input, output, latency, costs, prompts
  - Tools: Langfuse, Langsmith, many more

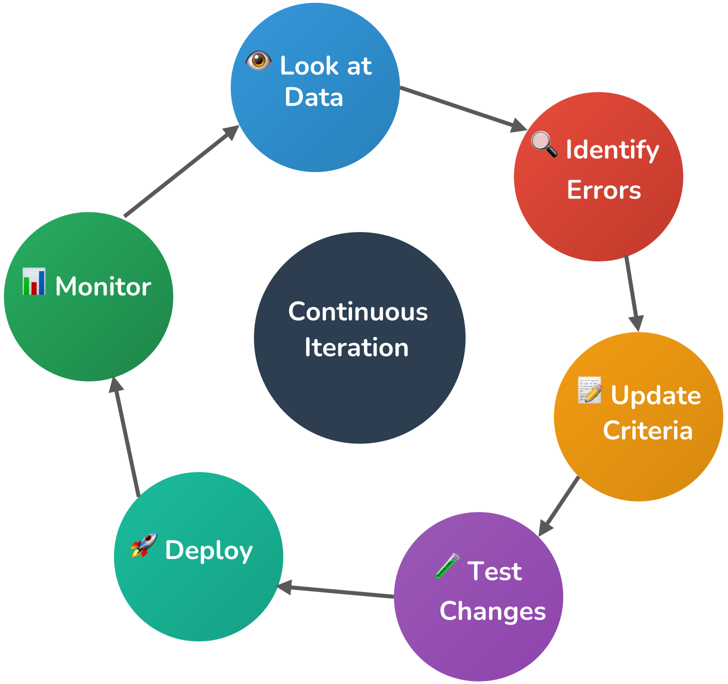
🏆 Rule Number 1
Always look at your traces



Production Cycle

Key Takeaways

- ✓ Look at your traces
- ✓ Start simple (binary pass/fail)
- ✓ Task-specific evals
- ✓ Validate validators
- ✓ Iterate continuously



You are the best judge of output quality. Don’t rely on numbers