Common Pitfalls when building Gen AI Applications - Chip Huyen

Date: __/__/____

1. Use GenAI when it is not needed (If all you have is a hammer, everything looks like a nail)
2. Confuse "Bad product" w/ "Bad AI" (UX is hard)
3. Start too Complex (Shiny object Syndrome, Be vigilant when adopting any abstractions)
4. Over-index on early Success
   - Easy to build a demo, hard to build a product

Common challenges:

- Accuracy / latency tradeoff
- Hard for agents to differentiate b/w similar tools
- Hard to Completely Understand Customers' intent
- Hard to Create Unit tests
- Reliability from API providers (timeouts)
- Safety & Compliance

5. Forgo human evaluation: Supplement LLM-as-a-judge w/ human eval. Improve System over time

#learnthesmarterway

Date: __/__/____

(6) Crowdsource Use Cases for GenAI

- Individuals might be biased toward the problems that immediately affect their day to day work instead of problems that might bring the highest ROI
- without an overall strategy that considers the big picture, it is easy to get sidetracked into a series of small, low impact apps and then come to the wrong conclusion that genAI has no ROI

---

12 Step checklist For AI model selection

yt vid

1. Cost
2. Modality: text, image, Audio, Video
3. Customisation options
4. Inference options: realtime, batch etc
5. Latency
6. Architecture
7. Performance benchmarks: leaderboards
8. Language
9. Size and Complexity
10. Ability to Scale
11. Compliance
12. Environment

#learnthesmarterway

Building a Gen AI Platform

– Chip Huyen

Date: __/__/____
Jan 18 2025

0. Model API : Query → LLM → Response

Step 1. Enhance Context

- The more relevant info in Context → less the model relies on internal knowledge → Better Ans ↓ hallucination
- RAG - Retrieval
  Generation
  Augmented
- Chunking : Splitting docs to manageable chunks. To decrease overall Context sent to LLM

Retrieval approaches :

① Term based : ex: Keyword Search
(Sparse vectors) BM25 (improves on TF-IDF)
(Dense vectors)

② Embedding based : data → Emb model → emb
(vector Search) Query Embedded, Compared w/ Emb data

③ Hybrid Search : ① + ②
(Useful for Exact words, names, acronyms etc)

#learnthesmarterway

Date: __/__/____

Sequential hybrid search: Cheap less precise retrievers → Some relevant docs → Costly more precise retrievers → Most relevant docs

- Exact position of relevant chunks do not matter as much as they do in traditional search, as long as they are present

Ensemble H+S: Combine different rankings together to get a final ranking

RAG but better Rerankers (James Briggs)

- We lose some information it is compressed to create a vector
- Say top-k is 3, but one relevant chunk is in 17th position, that chunk will not be sent to the LLM
- Say we ↑ it to 20, but then we are filling in the context window (limited) and 3-16 are irrelevant chunks
- So we want to retrieve all relevant chunks but not send all of them to LLM → So Reranking

#learnthesmarterway

Date: __/__/____

- Transformer is Computationally intensive. Cosine Similarity is not
- Generating embs for docs & storing them is doing the hard part first so that Cosine Similarity retrieval is easy (fast) at user runtime
- In Reranking, text is sent to LLM(docs, query) so info is uncompressed. So ranking is better but costly.

Ex: 1000 —retriever→ 25 —reranker→ 3
total docs retrieved docs most relevant docs

- These reranked chunks are sent to the LLM, to answer queries.

Back to Chip's article

RAG w/ tabular data

① Text 2 SQL: SQL created from User's query + db schema
② Run this SQL query
③ Pass results + query to LLM, get response

(Intermediate step to determine which tables for query may be required if all table schemas can't fit in context)

Date: __/__/____

Agentic RAG

- A workflow that can incorporate external actions is called agentic
- term based retrieval
  emb based retrieval
  SQL execution
  Web Search

  } Actions that a model can take to augment its context

Query Rewriting

- User might ask ambiguous questions, especially in follow up, as they might have provided context in previous Q.
- So last Q should be rewritten to ensure it makes sense on its own
- Typically done using AI models:
  "Given the following conversation, rewrite the last user input to reflect what the user is actually asking"
- If identity resolution is required (How about his wife), then it gets complicated. Rewriting model should not hallucinate

Date: __/__/____

Step 2: Put in Guardrails

- Protect users & developers, place it where there is potential for Failures.

Input G R
- Leaking private info to external APIs
- Model Jailbreaking

If query contains sensitive info
- Block Query
- Mask PII, send, unmask

→ Guardrails such that Insert, update, delete queries cannot be executed automatically

- Define out of scope topics & predefined phrases to reject user query

Output G R
- Output Quality Measurement
  - Empty responses
  - Malformatted responses
  - Toxic responses
  - Hallucinations
  - Sensitive info (trained or retrieved)
  - Brand Risk & other bad responses
- Failure management
  - Retry while balancing cost & latency
  - human fallback
    - Specific queries
    - Angry sentiment
    - After n turns

#learnthesmarterway

Date: __/__/____

Guardrail tradeoff

① Increased risk → Costlier than → Latency
So better to have guardrails

② OpenAI might not work in stream completion mode

③ Self hosting models → data not sent to 3rd party
But then, cannot rely on guardrails provided by 3rd party.

Step 3 — Add Model Router & Gateway

Router
- Specialized models: Better response, ↓ Cost
- Intent classification & Avoid out of scope queries
- next action predictor: Ask for clarification

Model gateway
- Unified Interface w/ models (Open & Closed) in secure manner
- Access control & Cost management
- Fallback Policies for rate limits / API failures
- Load balancing, logging, Analytics

ex: Portkey, litellm

Date: __/__/____

Step 4: Reduce Latency w/ Cache

Prompt Cache (at inference API)

- Many prompts have overlapping text segments (ex System prompt).
- This can be cached & reused so that it is processed just once (during first query)
- Supported by OpenAI, Claude and Gemini (called Context Caching)

Exact Cache

- System stores processed items for reuse later when exact items are requested
- Ex: Do not gen product summary if already exists
- Avoid redundant vector search by checking if incoming query is in vector search cache
- Useful for queries having multiple steps or time consuming actions

Semantic Cache

- Allows reuse of similar query results
- You should have reliable way to check if it is similar
- Might not be useful as many of its components are prone to failure

Date: __/__/____

Step 5: Add Complex logic & Write Actions

Complex logic
- Ability to Plan & decide what to do next
- loops & Conditionals

Write actions
- Can help automate Complex Flows
  ex: Customer outreach workflow
- prompt injection: Social Engineering on AI
- ex: Leaking Sensitive data from db, Corrupting data

Orchestration: Components & Pipelines

Observability

Metrics: model metrics: Accuracy, toxicity, hallucination retrieval Quality

System metrics: uptime, mem, throughput etc

Develop metrics to monitor different ways in which models Can fail

Logs Log everything

Traces requests execution Path through Components