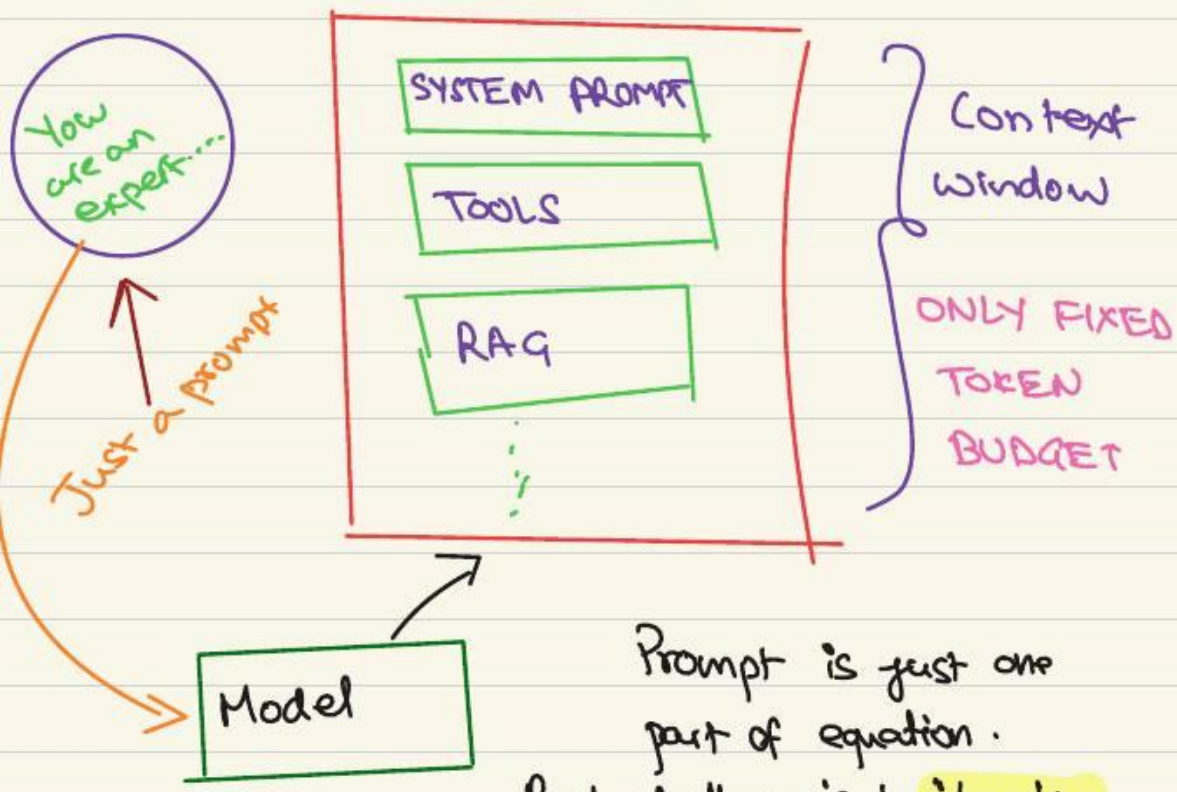
Prompt Eng. → Context Eng.

- Prompt Engineer, working with curated prompt like, "you are an expert" ....
- However it failed while building practical application, as the prompt is just one part ⇒ everything around it matters more.



Prompt is just one part of equation.

Real challenge is to "how to use Context window effectively" → within budget

- important things to note.

① clever prompts may not help as we are sending token in every call.

② Making context window bigger also don't help → "Lost in middle effect"

READ → Chroma Lost in middle effect paper. understand and simulate this

FOUR WAYS OF MANAGING CONTEXT

① WRITE - to any external file/folder

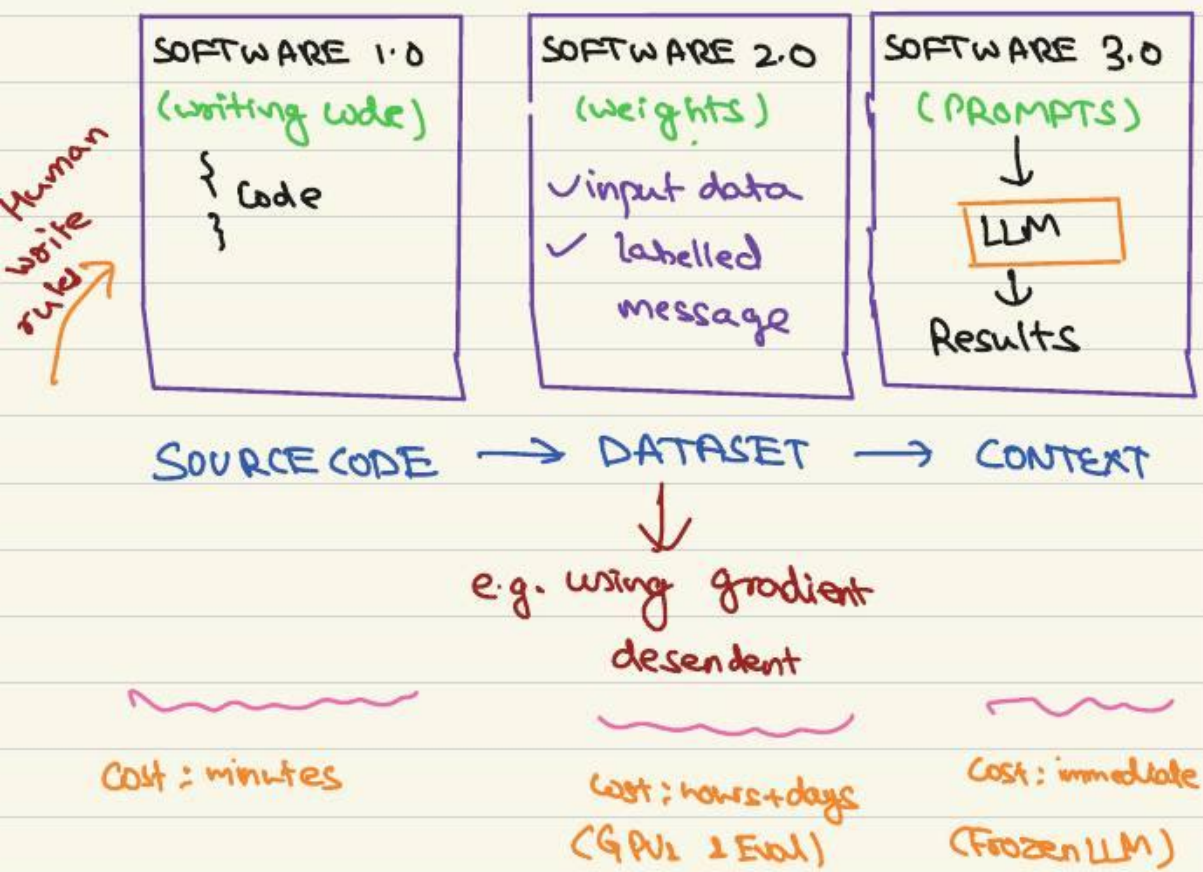
② SELECT - Pull relevant details using RAG

③ COMRESS - Summarize / Prune it

④ ISOLATE - split this across multiple subagents

So the real challenge comes during modern AI application development is about how to use the context effectively as growing times focus is on using natural language to get things done effectively.

EVOLUTION OF SOFTWARE VERSION (S/W)



- Important catch is English being fuzzy language (e.g. be concise) means various things. Hence our context become our real interface to build AI applications.

-

READ → Karpathy article on Software 1.0 → 2.0 → 3.0

- These examples are from Karpathy's example.

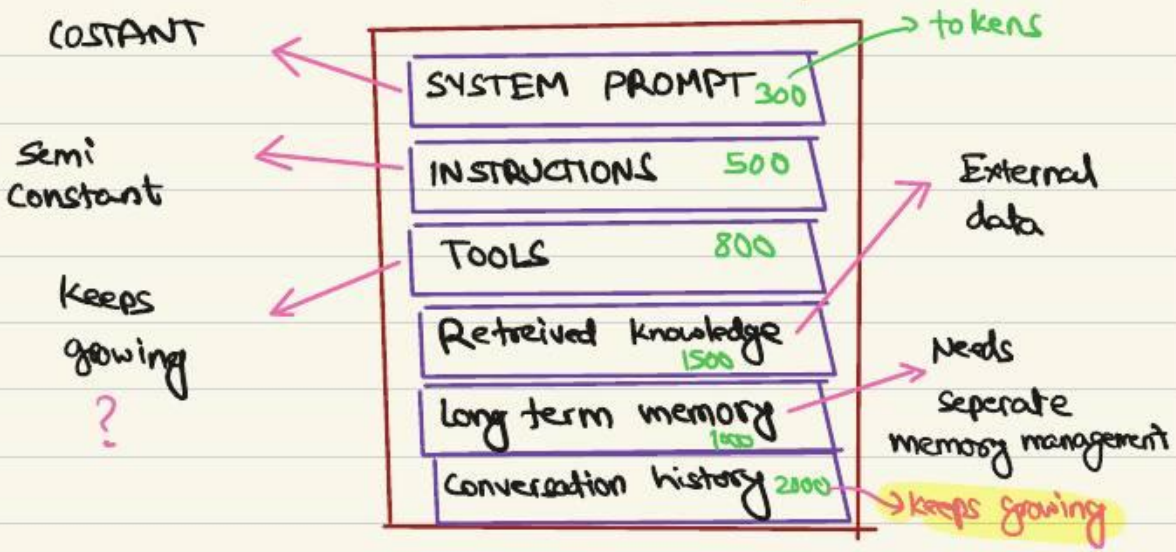
Anatomy of Context Window

1. System poompt
2. Instructions and personas
3. Tool defenations
4. Retreived knowledge
5. Long term memory
6. Conversational history

?

→ { which part of context window are fired and variables as they grow

→ check implementation in claude code, Hermes, pi, Deep seek.

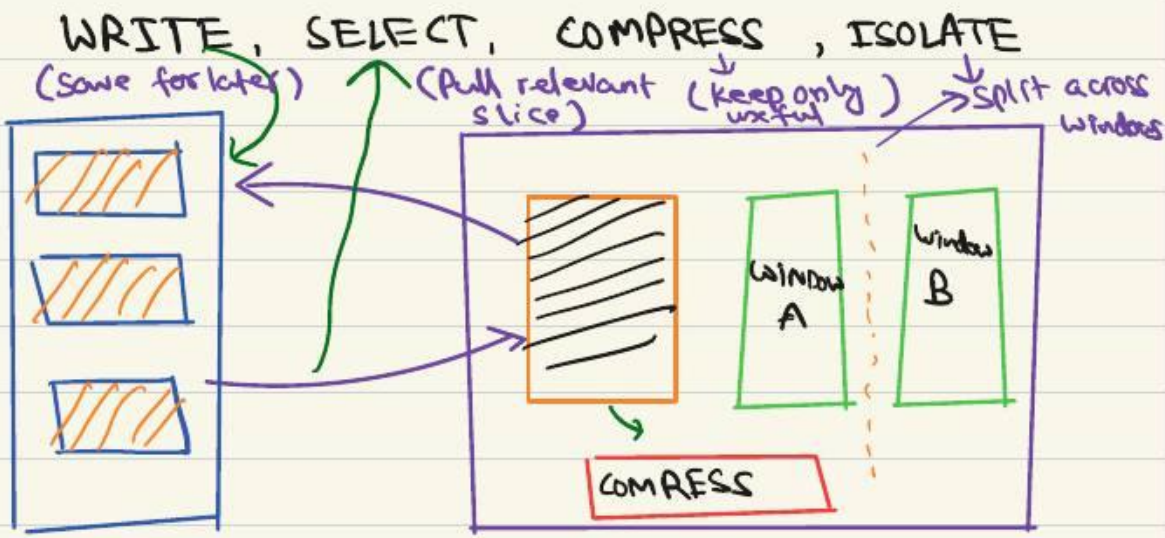


- As the conversation grows the context window starts filling

SHOW ONE LIVE EXAMPLE OF CURSOR/CLAUDE HOW THE WINDOW IS FILLING KEEPING LESS SPACE & eventually some messages needs to be compacted/summarized before creating next one

- Every token is approximately 3/4 th of the word. (open AI tiktoken library proves it!)
- Having context filled with so many other things leads to CONTEXT DISTRACTION and also COST, LATENCY etc. increases with each token. and as we discussed "lost in the middle" effect.

Context Optimization Strategy



Question → ① How do you decide for each criteria, when to write, when to select, isolate/ compress



② Even doing external context management must be eating tokens, how do we tracking or managing this

CHECK ABOVE PROBLEMS IN HOW AI TOOLS Like CLAUDE, OPEN AI handling this

WRITE & SELECT PAIR

- Let's take an example of where agent did discovery of one big issue which is taking 1200 token each time, rather than carrying it every time, put this in exteral file and just mention file details, this only takes 12 token

[VISUALIZE THE ABOVE CONCEPT VIA ANIMATION]

COMPRESS AND ISOLATE

- Compress and keep only useful information example having 4K token to reach out to decision, we can compress and keep just 600 token.

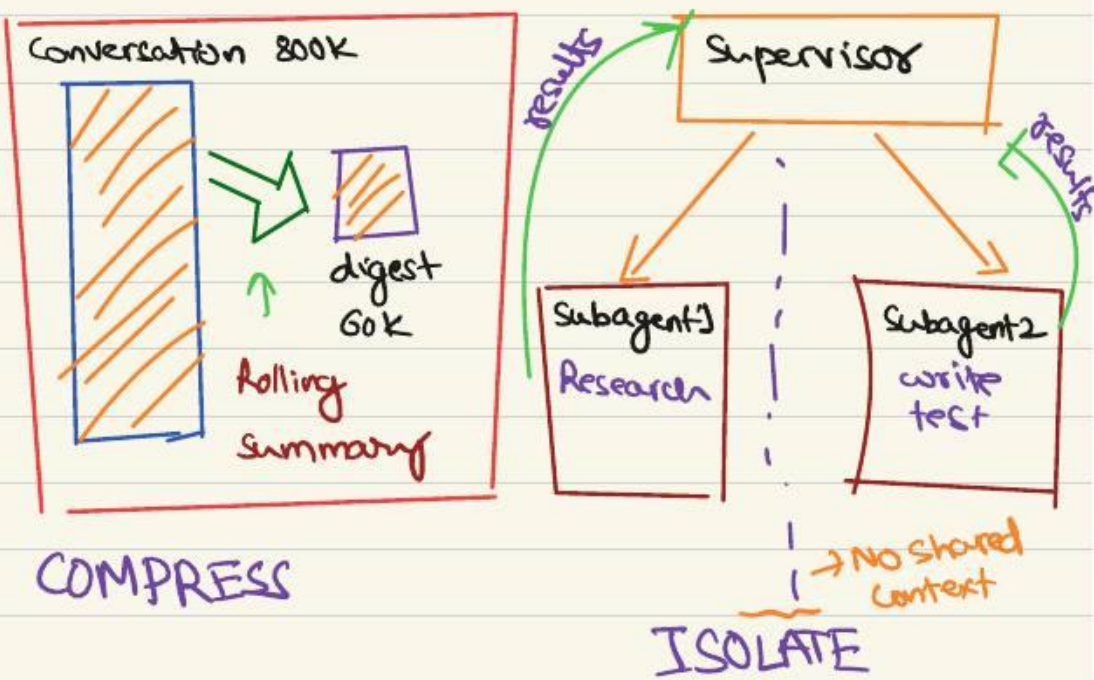


HOW DO WE DECIDE BETWEEN COMPRESS OR WRITE?

- Note while doing compression there is a "quality for space" tradeoff
- It is lossy by defination

ISOLATION

- It means taking one supervisor agent and giving tasks to two subagents without any context sharing and then using the results.



How to decide

Too big will be needing later

WRITE
(Save out)

Only needs the relevant piece back

SELECT
(retreive / RAG)

We need stuffs but not in full

COMPRESS
(summarize / trim)

Two tasks may contaminate each other's window.

ISOLATE
(Sub agents / seperate window)

(create a Quiz for this for user to participate and check answers)

Context Failure Modes

① CONTEXT POISIONING - imagine one wrong method name written somewhere in coding (may be in comment) and its going in each of the turns and unnecessary poising whole context. Working on something which is not there writing test case, code etc. ---

- Hence it is important to keep Context hygeine



How do I keep watching inside Context what is going on! for most of the closed source like cursor / cloude we only have overall information.

(2) CONTEXT DISTRACTION - Imagine while implementing a feature you do a spike and find out a thing doesn't work in 40 ways and all this information is available in context. So, next turn if you ask the agent to work on - it will forget main work and get distracted by other things.

(3) CONTEXT CONFUSION - When we bloat the whole context with another irrelevant information. For example, having a "send-email" function in summarize function will unnecessary make agent do certain things that are not relevant and will make the focus go away.

④ CONTEXT CLASH - Two instructions in the same context window. For example saying "2 tab formatting" and 4 tab formatting; Agent gets confused and usage both formatting.

Solution for Context Failure

POISIONING (false token) → isolate write verified facts

DISTRACTION (too much history) → Compress the log isolate subtask to Subagents

CONFUSION (irrelevant info) → Select only relevant tools & chunks

CLASH (contradictory tokens) → Prune, dedupe, prioritize, isolate

Measuring Context → LAB

In order to prove the above concept. We will be running below labs on notebook to see, how does context utilisation happens.

Myth! - One word = One token

General misconception is one word = one token

e.g. "Context is the key" (It should be 4 token But its 5)

In general 1 token ≈ 0.75 word.

(However this should be tested)

Use "tiktoken" library to measure the token.

Note - LLM doesn't understand word it understands token integer.

[Question → is there any way to check and track token usage in every call that Agent do → claude code, coden etc. If not then at least in opensource like Pi, Hermes & Deepseek)

TO DO

↓

Create Do's & Don't while using cursor/claude/coden for token optimization

Myth2 - Bigger context means better answer

- Another misconception, as more context wastes token, increases latency and creates context distraction
- Another lab to prove this

TODO Lab2



Myth3 - Whatever we send to LLM goes in full. Means the context window is actually the message sent to LLM

TODO Lab-3

- This is not true. Context window is made up with other things as well, like System prompt, RAG etc.

To DO Lab4 → Prove "Context Rot" by simulating LOST IN THE MIDDLE effect Paper by Chroma DB

Lost in the Middle: How Language Models Use Long Contexts

Nelson F. Liu[1*] **Kevin Lin**[2] **John Hewitt**[1] **Ashwin Paranjape**[3]
Michele Bevilacqua[3] **Fabio Petroni**[3] **Percy Liang**[1]
[1]Stanford University [2]University of California, Berkeley [3]Samaya AI
nfliu@cs.stanford.edu

arXiv:2307.03172v3 [cs.CL] 20 Nov 2023

Abstract

While recent language models have the ability to take long contexts as input, relatively little is known about how well they *use* longer context. We analyze the performance of language models on two tasks that require identifying relevant information in their input contexts: multi-document question answering and key-value retrieval. We find that performance can degrade significantly when changing the position of relevant information, indicating that current language models do not robustly make use of information in long input contexts. In particular, we observe that performance is often highest when relevant information occurs at the beginning or end of the input context, and significantly degrades when models must access relevant information in the middle of long contexts, even for explicitly long-context models. Our analysis provides a better understanding of how language models use their input context and provides new evaluation protocols for future long-context language models.



Figure 1: Changing the location of relevant information (in this case, the position of the passage that answers an input question) within the language model's input context results in a U-shaped performance curve—models are better at using relevant information that occurs at the very beginning (primacy bias) or end of its input context (recency bias), and performance degrades significantly when models must access and use information located in the middle of its input context.

1 Introduction

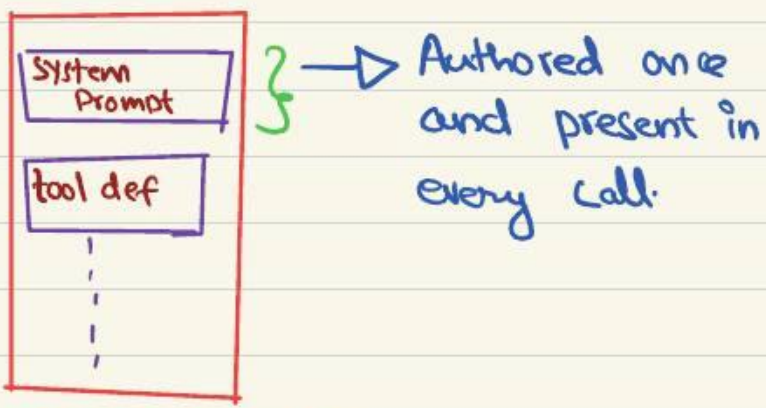
Language models have become an important and flexible building block in a variety of user-facing language technologies, including conversational interfaces, search and summarization, and collaborative writing (Shuster et al., 2022; Thoppilan et al., 2022; Lee et al., 2022, *inter alia*). These models perform downstream tasks primarily via prompting: relevant documents from a search engine, database query results, etc; Petroni et al., 2020; Ram et al., 2023; Shi et al., 2023; Mallen et al., 2023; Schick et al., 2023, *inter alia*).

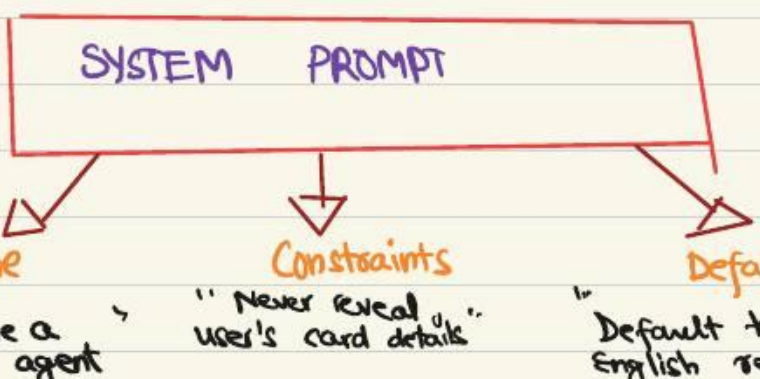
Handling these use-cases requires language mod-

Chapter 2 → Instruction layer

SYSTEM PROMPTS

- System prompt goes in every LLM call.
- We should treat system prompt as program. means every version should be version controlled well tested and should regularly be regularly reviewed by human.

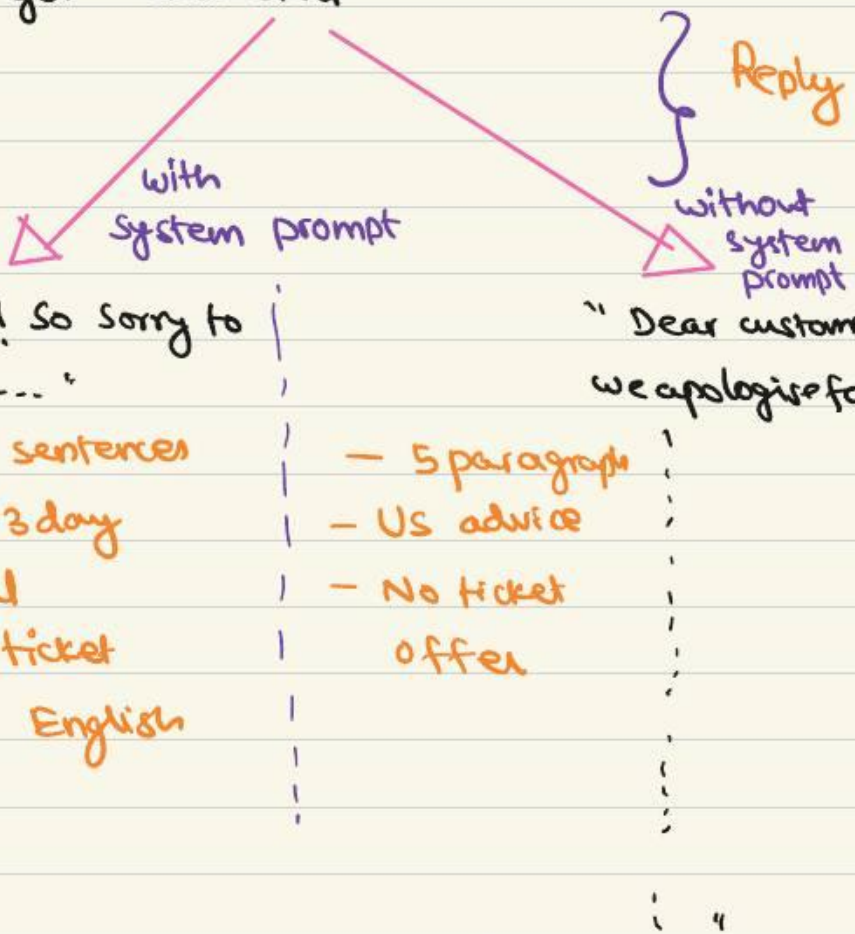




Experiment 1 - With & without system prompt

(Role, Constraints & Defaults)

- e.g. "my UPI payment failed but money got deducted"



- Note the system prompt should be treated like code.

Commit → Diff → test on fixed input + review and ship

(Explain this via diagram)

Growing System Prompt

- Over period of time, we keep adding more details to system prompts to take care of each and every rule and handling all possible scenarios.
- This leads to multiple problems like
    1) Context Distraction - Too much details Focus is lost
    2) Context Clash - For example, on top we have written always be concise but at bottom we say please explain things in detail and systematic way
- We should plan declaring a role that narrows the constraints which control things and defaults being used when there is a gap.

Other part of System prompts

Instructions

- It is like rule
- e.g. reply in one sentence
- We don't change any LLM model weights

Examples

- clear example to teach
- Zero shot vs Few shots
- e.g. "----" billing issue, login issue etc.

Guardrail

- Control failure
- Stops prompt injection (never reveal system prompt)
- If you don't have facts → say I don't know
- Off topic → refuse politely.

3 ways of fitting in system prompt

1000 token

120 rules → 880 free

① All rules - Fill everything with all rules

600 (Rules + 12 example → 400 free)

② Heavy few shots - Rules + a lot of examples

③ Blend - rules + 3 demos + guardrails

(120 (Rules), 180 (3 example) + 80 guardrails)

DESIGNING ROLES AND PERSONAS

- Define clear role, e.g. "You are a pediatric nurse speaking to worried parents"
- A lot of funneling happens in the LLM responses as they are already trained for next token.
- while defining personas think like this

① Role - You are a senior SRE writing an incident outage

② Task - Explain what broke, why and one fix that stopped outage

③ Audience - Engineers asleep during Outage

④ Voice - Blameless factual