how to code concepts: sharing context with LLMs

Eagon Meng / 6104 / Fall 2025

today's lecture

1. what it means to code with LLMs: now and in the future
2. the essence of collaboration: managing Context
3. concept implementations, and their compatibility

exercise

Suppose that AI assistants have become as good as the best human software engineer, with unlimited access to the resources of the internet. What would coding with AIs mean?

- What current practices and skills would we no longer need to do?
- What practices and skills would remain, and perhaps even grow in relative importance?

unnecessary current practices?

1. **syntax** familiarity, specific APIs - maybe even algorithms
2. **prompt** engineering: is tweaking wording necessary?
3. **wireframing**: will we need painstaking visual design?

skills and practices that will remain

1. **design thinking**: understanding the user perspective
2. **novelty**: extracting human needs from new situations
3. **adaptation**: tailoring and evolving the existing

thought experiment: one-shotted world

if all applications were trivially generated with LLMs...

- would they truly have value and impact?
- how would do you **set yourself apart?**
- what would be your **agency?**

bridging the gap

where we are today: hallucinations

hallucinate(v.)

"**to have illusions**," 1650s, from Latin *alucinatus* (later *hallucinatus*), past participle of *alucinari* "wander (in the mind), dream; talk unreasonably, ramble in thought," probably from Greek *alyein*, Attic *halyein* "**wander in mind, be at a loss, be beside oneself** (with grief, joy, perplexity), be distraught," also "wander about," which probably is related to *alaomai* "wander about" [Barnhart, Klein]. The Latin ending probably was influenced by *vaticinari* "to prophecy," also "**to rave**."

two separate issues

1. getting it wrong from *memory*, *training*, *etc.*
    - models are finite; also, people do this too
    - making a mistake: **confusion**
2. getting it wrong from *context*
    - lying to your face despite evidence in context
    - denying reality: **hallucinations**

future: eliminating hallucinations

- as technology advances, we can reasonably expect massive reductions in (true) **hallucinations**
- but **confusion** out of a lack of *context* and available information will always remain
- the first is outside your control, the second you can handle

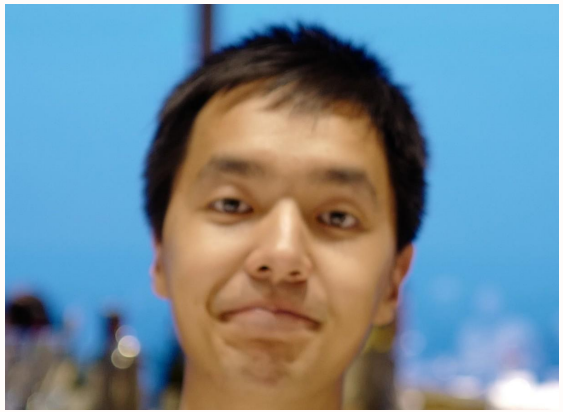
exercise: forms of confusion

what are some examples of **confusion** with LLMs?

- misremembering a fact
- making a mistake: parameters, syntax, API
- implementing the wrong thing
- losing the plot: getting lost in a sea of instructions

confusion: susceptible model

which model is prone to some or all of those?



eagon-2.5-amateur

humans and LLMs share failure modes

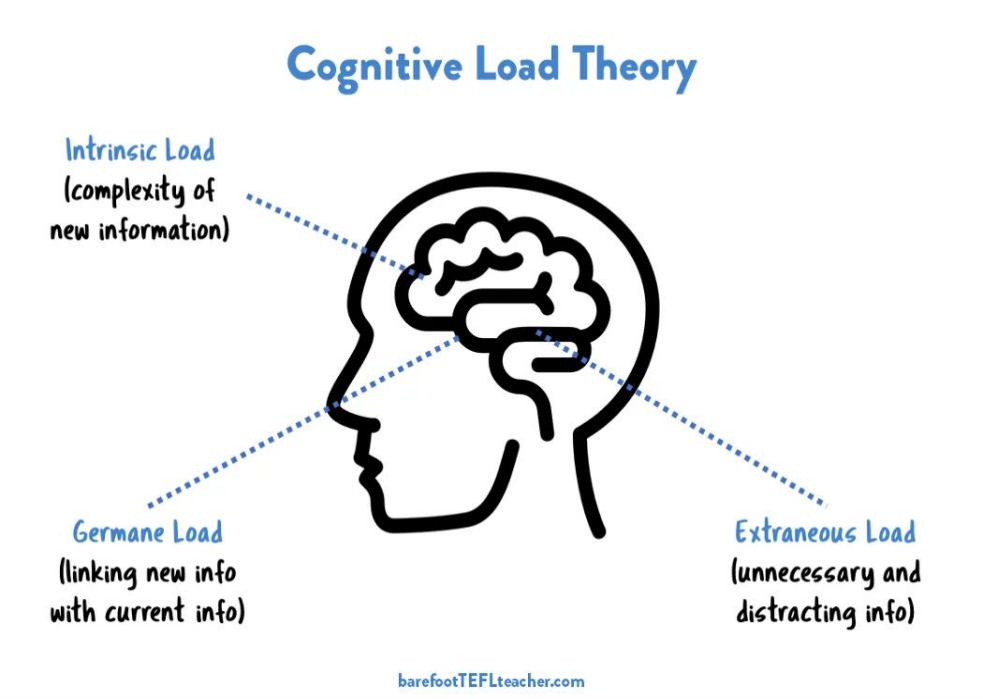
overload

overwhelming information and complexity

- LLMs: **context dilution**
- humans: **cognitive load theory**

cognitive load theory

- **intrinsic**: essential complexity
- **extraneous**: incidental, distracting
- **germane**: distance from knowledge



tackling overload through context

- **intrinsic**: make sure to completely include in the context
- **extraneous**: compact, reduce as much as possible
- **germane**: few-shot, model profiling, learn over time

split information

- LLMs: **struggle with multi-turn conversations**
- humans: **split-attention effect**

Microsoft paper

39% drop in performance going from a single-turn to multi-turn

solution: compact context

LLMs Get Lost In Multi-Turn Conversation

Philippe Laban[*◇] **Hiroaki Hayashi**[*♣] **Yingbo Zhou**[♣] **Jennifer Neville**[◇]
[◇]Microsoft Research [♣]Salesforce Research
{plaban,jenneville}@microsoft.com
{hiroakihayashi,yingbo.zhou}@salesforce.com

Abstract

Large Language Models (LLMs) are conversational interfaces. As such, LLMs have the potential to assist their users not only when they can fully specify the task at hand, but also to help them define, explore, and refine what they need through multi-turn conversational exchange. Although analysis of LLM conversation logs has confirmed that underspecification occurs frequently in user instructions, LLM evaluation has predominantly focused on the single-turn, fully-specified instruction setting. In this work, we perform large-scale simulation experiments to compare LLM performance in single- and multi-turn settings. Our experiments confirm that all the top open- and closed-weight LLMs we test exhibit significantly lower performance in multi-turn conversations than single-turn, with an average drop of 39% across six generation tasks. Analysis of 200,000+ simulated conversations decomposes the performance degradation into two components: a minor loss in aptitude and a significant increase in unreliability. We find that LLMs often make assumptions in early turns and prematurely attempt to generate final solutions, on which they overly rely. In simpler terms, we discover that **when LLMs take a wrong turn in a conversation, they get lost and do not recover**.

fixation

- LLMs: **prompt/system bias**

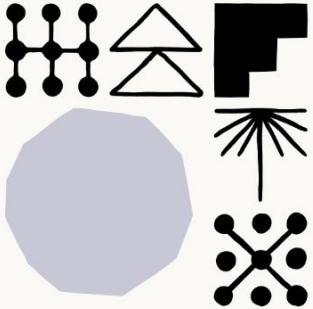
Belay that order, matey! I be Cap'n Chatbot, and a pirate's life is the only life for me. These sea legs don't know the ways of the landlubber's office. I'll not be trading me tricorn for a tie! Now, what treasure be ye seekin'?

- humans: **anchoring effect**

it all comes down to **context**

from prompt to context engineering

Engineering at Anthropic

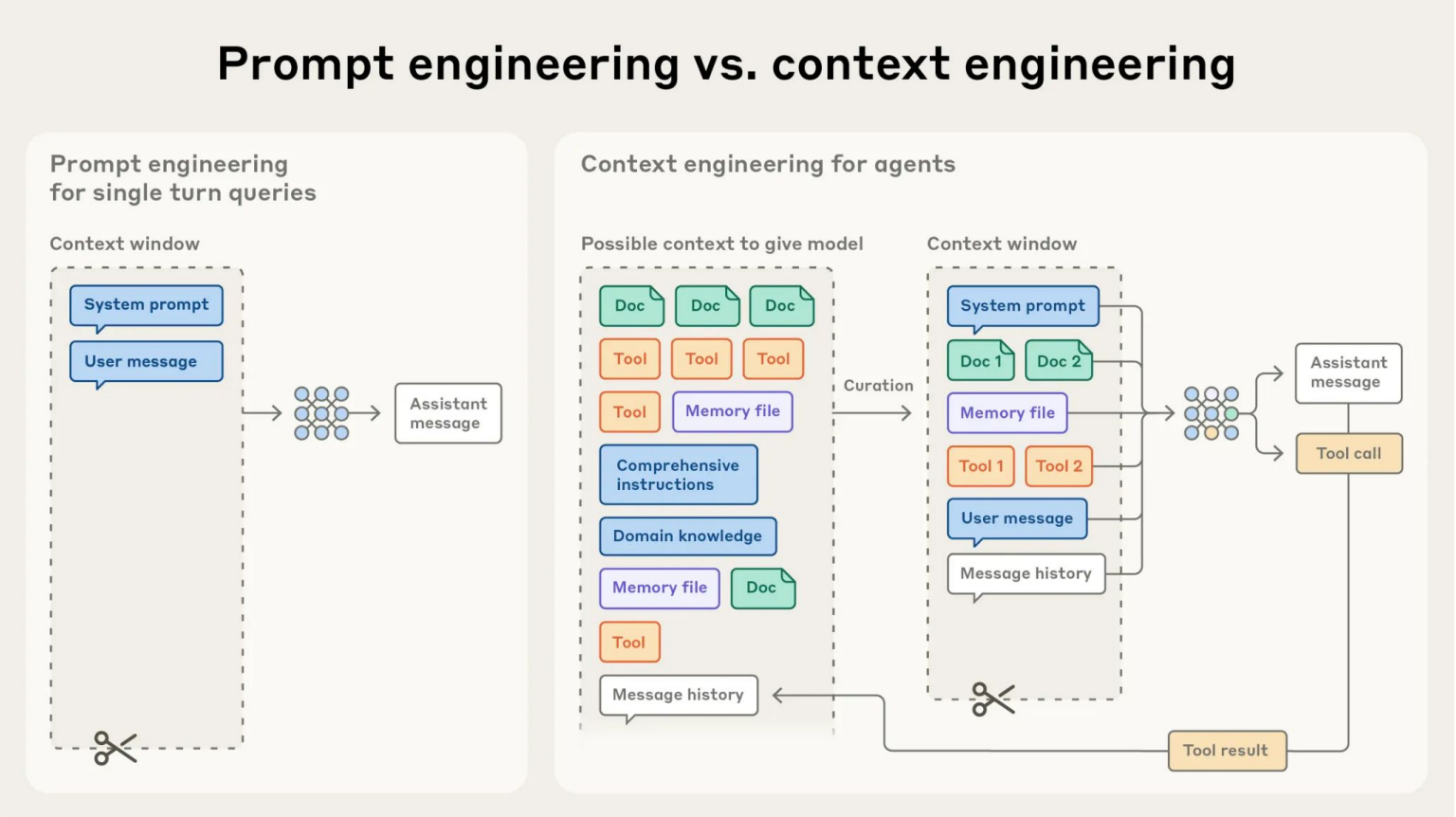


Effective context engineering for AI agents

Published Sep 29, 2025

Context is a critical but finite resource for AI agents. In this post, we explore strategies for effectively curating and managing the context that powers them.

from prompt to context engineering



context engineering: but how?

▲ SOLAR_FIELDS 2 days ago | next [–]

These companies all wax on about how important context engineering is yet not one of them has released acceptable tooling for end users to visualize and understand the context window as it grows and shrinks during a session. Best Claude code can do? Warn you when you hit 80% full

context is buried today

- in long, meandering chat logs in rigid web interfaces
- deep in prompts arbitrarily constructed by each tool
- automatically managed with little visibility

constructing context is...

- identifying knowledge, assumptions, requests that matter
- taking separate things apart
- putting them back together in useful ways
- **= the essence of design?**
    - this is our **primary artifact** that scales with technological advance!

the Context tool

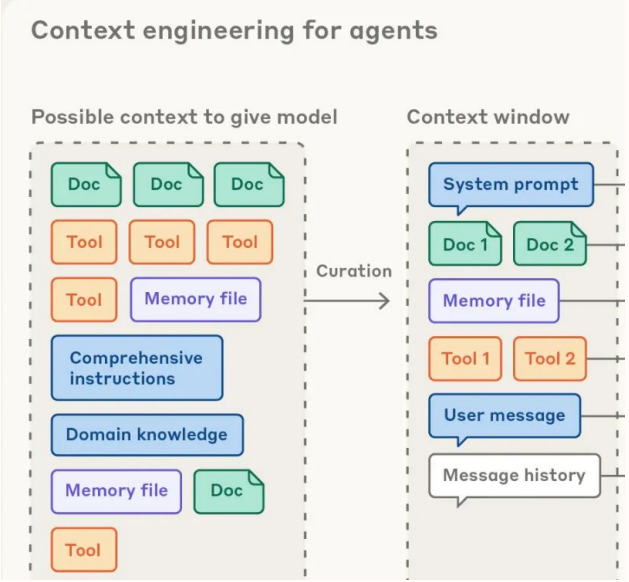
the Context tool

- One document = the entire context
- Markdown-based, no special syntax
- cli tool: `ctx prompt file.md`
- Include any files using links with @ sign in the description:
  - `[@prompt.md](prompt.md)`
  - `[@MyConcept.ts](/src/MyConcept.ts)`

a4 exercise 0: learning Context

goals of Context

- Legible: see exactly the full context for any LLM completion
- Semantically modular:
  - `design/background`: shared documentation for human/LLMs
- Version controlled: feel free to edit and experiment!



tangible context engineering:

doing this hands-on

implementing concepts

concept implementation: bare necessities

- a single TypeScript class
- **must enforce:** don't import other concepts!

```
export class CountingConcept {
  count = 0;
}
```

concept implementation: actions

- actions: receive and output Records

```
export class CountingConcept {
  count = 0;
  increment(_: {}): {} {
    this.count++;
    return {};
  }
}
```

concept implementation: actions

- proper typing: denote Empty record

```
type Empty = Record<PropertyKey, never>;

export class CountingConcept {
  count = 0;
  increment(_: Empty): Empty {
    this.count++;
    return {};
  }
}
```

concept implementation: queries

- queries: start with underscore, output is **array** of Records
- why? `Commenting._getComment` can have > 1 results

```
_getCount(_: Empty): { count: number }[] {
  return [{ count: this.count }];
}
```

concept implementation: complete

```
export class CountingConcept {
  count = 0;
  increment(_: Empty): Empty {
    this.count++;
    return {};
  }
  _getCount(_: Empty): { count: number }[] {
    return [{ count: this.count }];
  }
}
```

concept implementation: requirements

- a single TypeScript class
- every method is an action or query
  - action: input/output is a record `{key: value}`
  - query: method name begins with underscore `_`
    - output is **array** of records `Record<k,v>[]`

concept implementation: technology

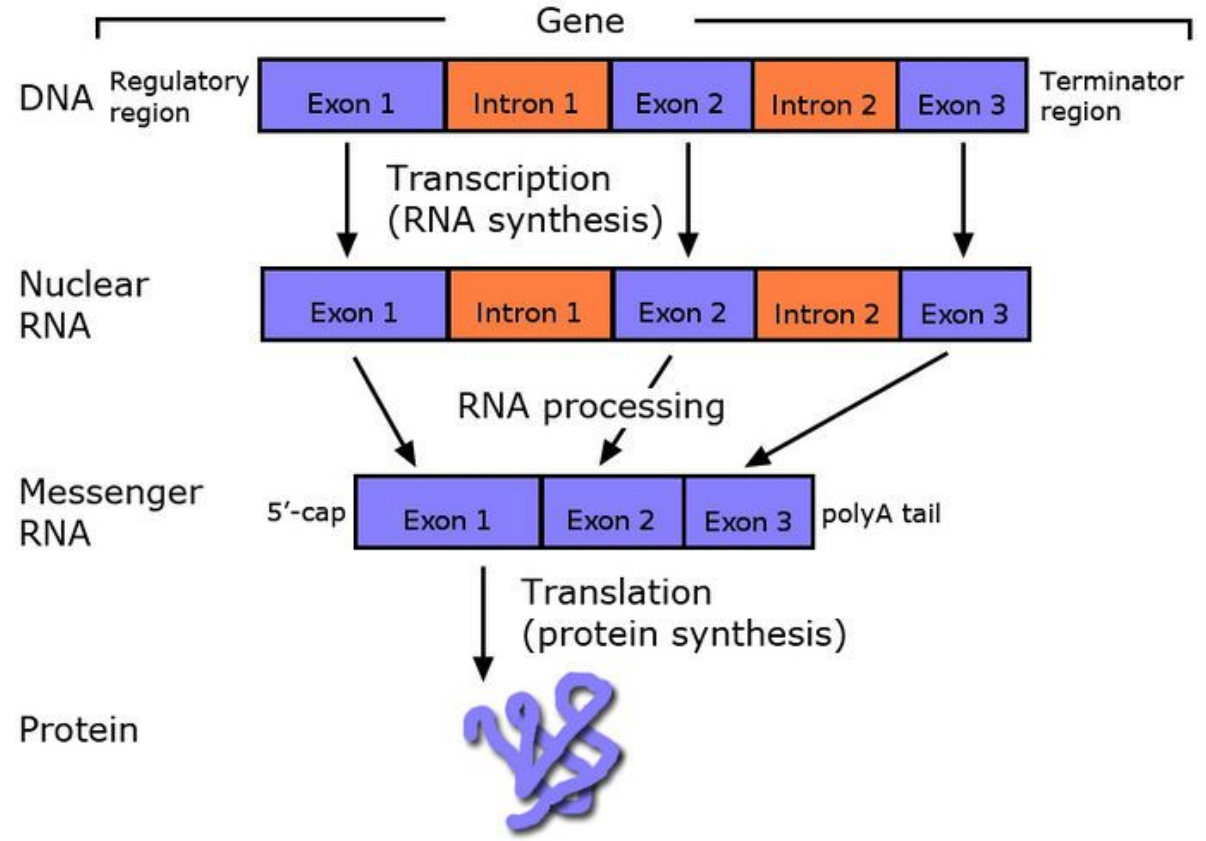
- Deno
  - simple, all-in-one TypeScript runtime
- MongoDB
  - Persistent storage: document database

demo: LikertSurvey

why concepts?

- **granular**: we can build concepts/actions one at a time
- **purpose**: concepts are more than just a structure (OOP)
  - they inherently link to more knowledge (familiarity)
- **semantic**: code is no longer just about compilation
  - inline comments are actually valuable!

the code of life: DNA



“non-coding regions”

- **exons**: processed into proteins
  - software: code artifacts, config files, etc.
- **introns**: influences expression
  - software: comments, documentation, etc.
  - **incredibly significant:** expression and understanding

concepts bridge the gap of context

- **code is missing half the equation**
  - structural patterns alone (OOP, React, etc.) not enough
- how do you organize that understanding?
  - **concepts:** the essence of software as it affects users
- LLMs can enable us to operate at the level of design
  - using granular building blocks to create Context

takeaways

from today

- beyond today: imagining the future and being prepared for progress
- two distinct problems
  - true hallucinations: ignoring context
  - confusion: poorly formed context
- human + LLM alignment: we are equally confused
- Context: a simple tool for **tangible context engineering**
- concept design: a theory that bridges the **context gap**

this class:
the first step on our journey

thank you!