Context Engineering

1. Introduction

Context Engineering is the practice of **intentionally selecting, organizing, and maintaining the most relevant information** provided to a Large Language Model (LLM) so that it produces accurate, consistent, and useful responses—especially across **multi-turn conversations**.

From a developer's perspective, context engineering is similar to:

- Managing application state
- Designing API request payloads
- Handling session memory
- Curating data inputs

Without proper context, LLMs:

- Forget earlier instructions
- Hallucinate details
- Produce inconsistent outputs

2. Why Context Engineering Matters

2.1 Problems Without Proper Context

Issue	**Example**
Loss of intent	Model forgets constraints
Inconsistent answers	Different responses to same question
Hallucinations	Assumes missing data

Issue	Example
Token waste	Too much irrelevant text

2.2 Benefits of Good Context Engineering

- Higher accuracy
- Lower token usage
- Stable multi-turn conversations
- Production-ready outputs
- Predictable system behavior

3. What Constitutes “Context” in LLM Interactions

3.1 Types of Context

Context Type	Description
System context	Role, rules, behavior
Conversation history	Previous user/assistant messages
Task context	Current objective
Domain context	Business rules, terminology
Data context	Inputs, schemas, examples

3.2 Context vs Prompt

Concept	Meaning
Prompt	Immediate instruction
Context	Supporting information guiding interpretation

4. Core Principles of Context Engineering

4.1 Relevance Over Volume

More context ≠ better output

Relevant context = better output

Bad Practice

Include entire documentation in every prompt

Good Practice

Include only the API contract and constraints needed

4.2 Structured Context Beats Free Text

Always organize context into **clear sections**.

Example

```
System Rules:

Domain Rules:

Input Data:

Expected Output:
```

4.3 Stability First, Flexibility Second

- Keep **system rules stable**
- Update **task context dynamically**

---

5. Structuring Context for Single-Turn Tasks

5.1 Recommended Context Structure

```
System Role

Domain Context

Task Instruction

Input Data

Constraints

Output Format
```

5.2 Example: Code Generation

```
System:

You are a senior Java backend developer.



Domain Context:

This is a Spring Boot application.



Task:

Generate a service method to calculate order total.



Constraints:

- Java 17

- No external libraries



Output:

Code only.
```

6. Context Engineering for Multi-Turn Conversations

6.1 The Multi-Turn Challenge

LLMs do not have long-term memory by default. Context must be **re-supplied or summarized**.

6.2 Conversation Context Layers

Layer	**Purpose**
Static system context	Behavior and rules
Session context	User goal
Turn context	Latest input
Memory summary	Key decisions so far

6.3 Example: Multi-Turn Development Assistant

Turn 1

User: Design a REST API for order management

Turn 2

User: Add authentication

Without context → model redesigns entire API

With context → model extends previous design

7. Selective Context Injection

7.1 What to Inject

Include	Exclude
Key rules	Redundant text
Decisions made	Raw logs
Schemas	Unused APIs

7.2 Example: Bug Fixing

Context:

- Language: Java

- Framework: Spring Boot

- Error: NullPointerException in OrderService

8. Context Engineering vs Prompt Engineering

Aspect	Prompt Engineering	Context Engineering
Focus	Instructions	Information selection
Scope	Single request	Multi-turn
Goal	Output quality	Consistency

Conclusion

Context Engineering is essential for building **reliable, scalable, and production-grade LLM applications**. By carefully selecting and structuring relevant information, developers can ensure that models maintain **continuity, accuracy, and intent**, even across long, multi-turn interactions.

When done correctly, context engineering transforms LLMs from reactive chatbots into **state-aware intelligent systems**.