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

---

2. Why Context Engineering Matters

2.1 Problems Without Proper Context

- Loss of intent: Model forgets constraints
- Inconsistent answers: Different responses to same question
- Hallucinations: Assumes missing data
- 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

---

3. What Constitutes "Context" in LLM Interactions

Types of Context:

- 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

---

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

```
System Rules:
Domain Rules:
Input Data:
Expected Output:
```

---

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 Python backend developer.

Domain Context:
This is a Flask/FastAPI application.

Task:
Generate a service function to calculate order total.

Constraints:
- Python 3.11+
- 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

- 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

Include: Key rules, Decisions made, Schemas

Exclude: Redundant text, Raw logs, Unused APIs

Example: Bug Fixing

```
Context:
- Language: Python
- Framework: FastAPI
- Error: TypeError in order_service.py
```

---

8. Context Engineering vs Prompt Engineering

- **Prompt Engineering:** Focus on instructions, single request scope

- **Context Engineering:** Focus on information selection, multi-turn scope

---

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.