Context Engineering 101

Stop Writing Prompts, Start Engineering Context

A Practical Guide for SREs & DevOps Engineers

What is Context Engineering?

The Definition

The practice of providing AI assistants with the right information, in the right format.

It's about making the agent's job easy by reducing ambiguity through intentional structure.

Not Just Clever Prompts

- Focus on **structure** and organization.
- Let the **environment** tell the story.
- Create searchable **ground truth**.
- Build systems that give AI what it needs automatically.

Course Overview

1. Core Concepts

Four context strategies and key principles like Space Jam Theory.

2. Filesystem Ops

Structure as context architecture and consistent naming.

3. MCP Servers

Connecting to external APIs, databases, and cloud inventories.

4. Orchestration

Managing multiple specialized agents and patterns.

Module 1: Core Concepts

Strategies & Principles

The Four Context Strategies

SELECT

Choosing the right files. Don't dump the whole repo. Curate the context.

WRITE

Creating **Agent Guidance Documentation** to guide specific behavior and rules.

ISOLATE

Using separate chats/threads for separate concerns to prevent context bleed.

COMPRESS

Summarizing long chats into **handoff prompts** or state files to reset context.

Space Jam Theory

"If you can dream it, you can do it"

Don't self-limit based on perceived complexity.

- Express uncertainty: *"I've never used Terraform, but I need to..."*
- You bring domain knowledge; AI brings generation.
- Amplify your expertise, don't replace it.

"AI can read production.
Only you execute against production."

— THE GOLDEN RULE OF SRE AI

"AI wrote it" is never an excuse. You own exactly what runs under your credentials.

Meeseeks Theory: "Can Do!"

The "Problem" isn't Hallucination

The problem is that AI will enthusiastically do **precisely** what you ask, even if it's useless.

Clear context = Useful results.
Vague context = Technically correct but unhelpful.

Vague vs. Specific

- ✗ "Fix the config"
- ✓ "Update resource limits in /helm/api/values.yaml"
- ✗ "Update the script"
- ✓ "Add dry-run flag to backup-postgres.sh"

Module 2: Filesystem Organization

Your structure is your context

Filesystem IS Context

```
~/company/stuff

stuff/
├──── file1.yaml
├──── script.sh
├──── thing.tf
├──── backup-old.yaml
└──── test.py

# Problem: No context, unclear purpose.
```

```
~/company/SRE

SRE/
├──── helm/
│   └──── charts/user-api/
├──── terraform/
│   └──── azure-infrastructure/
├──── scripts/
│   └──── backups/
└──── docs/

# Benefit: Implicit context from location.
```

Best Practices: Navigation & Naming

Guided Navigation

Don't just `cd` around.

Use **Agent Guidance Documentation** to instruct the harness on where to look.

"When asking about database schemas, automatically check `src/db/schema.prisma`."

Descriptive Naming

- ✅ `backup-postgres-prod.sh`
- ❌ `script.sh`
- ✅ `azure-prod-networking.tf`
- ❌ `main.tf`

Good names prevent questions before they're asked.

Module 3: MCP Servers & Skills

Connecting to the outside world

Context on Demand

Servers vs. Skills

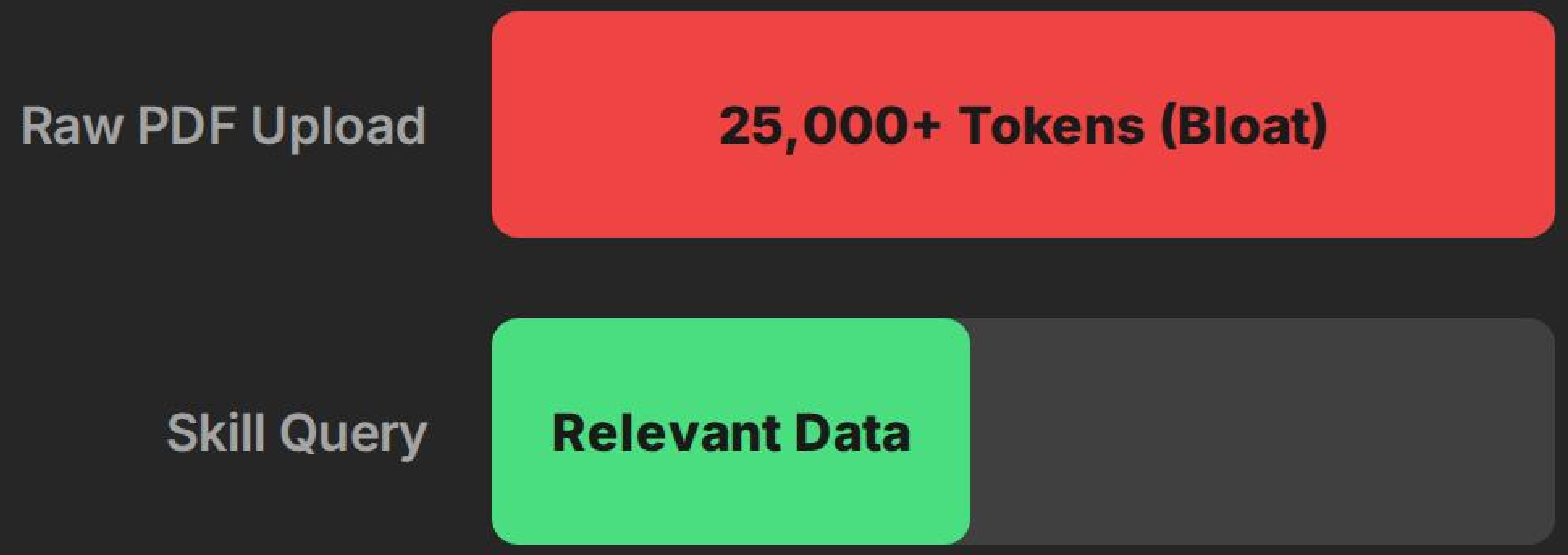
Servers are the pipe. **Skills** are the tools.

Skills enable **Lazy-Loaded Context**. They preserve context until needed.

Example: The Doc Parser

- **The Problem:** A technical manual (25k+ tokens) destroys your context window.
- **The Skill:** Agent converts file to embeddings on-the-fly.
- **The Win:** It retrieves *only* the specific paragraphs needed to answer.

Token Usage



Skills keep the context window clean for reasoning.

Module 4: Multi-Tab Orchestration

Managing specialized agents

One Tab, One Job

Orchestrator

Maintains the master plan. Generates **handoff prompts**.

Implementation (Green)

Execution. "Take this handoff and build the module."

Investigation

Log analysis. Safe.

Reference

Docs & Standards.

Urgent

Incidents.

Module 5: Patterns & Anti-Patterns

Productivity & Safety

Creation vs. Verification

Manual Workflow
Creation (4h)
Verify (30m)
AI Workflow
Gen
Verify (30m)

Result: 4-7x productivity boost. You shift from "Manual Creator" to "Expert Verifier".

Core Safety Patterns

1. Dry-Run Everything

Mandatory for operational scripts. Output must show what *would* happen.

```
./cleanup.sh --dry-run
```

2. Read vs. Execute

- ✅ AI can **READ** (Logs, Configs)
- ✅ AI can **GENERATE** (Scripts, Fixes)
- ❌ AI should NOT **EXECUTE**

You execute after verification.

3. Principle of Least Privilege

Give the Agent **Read-Only** access to cloud resources where possible.

If it needs to run plans (e.g., Terraform plan), ensure it cannot `apply` without human intervention.

4. The Git Safety Net

Never let AI commit directly to `main`.

All generated code goes into a PR. The Diff is your ultimate review tool.

Key Takeaways

- → **One Agent, One Job:** Use multiple tabs for isolated context.
- → **Filesystem is Context:** Organize logically, start in the right place.
- → **Tokens are Cheap, Ambiguity is Expensive:** Don't clutter context.
- → **Accountability:** AI generates, **only you** verify. No excuses.
- → **Context Engineering:** Making the AI's job easy.

?

Questions?

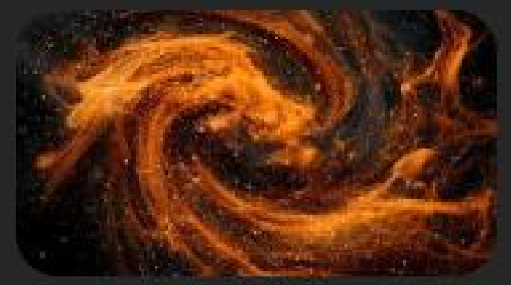
Context Engineering 101

Start applying these principles today.

Image Sources

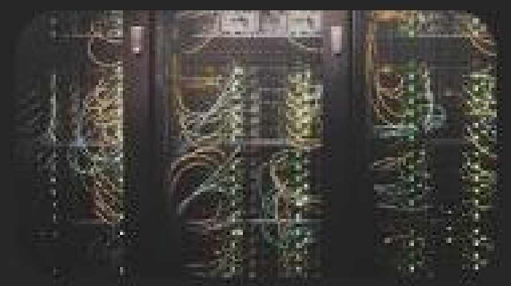
https://img.freepik.com/premium-psd/red-black-contrast-abstract-technology-background-generative-ai_271628-1835.jpg

Source: www.freepik.com



https://png.pngtree.com/background/20250524/original/pngtree-orange-and-black-cosmic-nebula-swirl-with-glitter-galaxy-space-abstract-picture-image_16572384.jpg

Source: pngtree.com



https://www.trgdatacenters.com/wp-content/uploads/2021/07/taylor-vick-M5tzZtFCOfs-unsplash-scaled.jpg

Source: www.trgdatacenters.com