🧠 Context Engineering vs Prompt Engineering vs RAG

Understanding the Three Pillars of AI Intelligence

🎯 Quick Comparison

Aspect	Prompt Engineering	Context Engineering	RAG (Retrieval Augmented Generation)
What	Design AI personality & behavior	Provide relevant info for THIS conversation	Search knowledge base for facts
When	Once, at design time	Every message	When user asks questions
Who	Orchestrators design	System provides automatically	System searches automatically
Where	System prompt	Conversation history	External knowledge base
Example	"You are a friendly agent"	"User said age is 28"	"Collision coverage definition"

📊 Visual Comparison

```
┌──────────────────────────────────────────────────────────────┐
│                        USER MESSAGE                          │
│                 "What is collision coverage?"                │
└──────────────────────────────────────────────────────────────┘
                            ↓
          ┌─────────────────┼─────────────────┐
          ↓                 ↓                 ↓
┌───────────────┐  ┌─────────────────┐  ┌──────────────┐
│   PROMPT      │  │   CONTEXT       │  │     RAG      │
│ ENGINEERING   │  │ ENGINEERING     │  │    SEARCH    │
└───────────────┘  └─────────────────┘  └──────────────┘
          ↓                 ↓                 ↓
┌───────────────┐  ┌─────────────────┐  ┌──────────────┐
│ "You are a    │  │ Previous msgs:  │  │ Knowledge:   │
│  friendly     │  │ - User: age 28  │  │ "Collision   │
│  insurance    │  │ - User: Honda   │  │  coverage    │
│  agent"       │  │ - AI: asked for │  │  pays for    │
│               │  │   history       │  │  damage..."  │
└───────────────┘  └─────────────────┘  └──────────────┘
          └─────────────────┬─────────────────┘
                            ↓
                   ┌───────────────┐
                   │  GEMINI LLM   │
                   │  Combines     │
                   │  all three    │
                   └───────────────┘
```

```
                    ↓
              ┌───────────────┐
              │   RESPONSE    │
              │ "Collision    │
              │  coverage..." │
              └───────────────┘
```

1️⃣ PROMPT ENGINEERING

What It Is

Designing the AI's **personality, role, and behavior** that stays constant across all conversations.

Analogy

Like hiring an employee and giving them a job description.

In the Workshop

Part 2 (15 min): Orchestrators design the agent's personality

Example Prompt:

```
You are "Alex", an expert insurance agent powered by AI.

Your personality:
- Friendly and professional
- Patient and helpful
- Explains complex terms simply

Your role:
1. Help customers get accurate quotes
2. Answer questions about coverage
3. Gather required information

Guidelines:
- Ask 1-2 questions at a time
- Explain clearly when asked
- Calculate quote when ready
```

Code Location

`backend/system_prompt.py`

```
INSURANCE_AGENT_PROMPT = """
You are "Alex", an expert insurance agent...
"""
```

When It's Used

- ✅ **Once** when agent starts
- ✅ **Same for all users**
- ✅ **Defines WHO the agent is**

Orchestrator Activity

1. Open AI Studio
2. Design personality
3. Test with conversations
4. Refine based on results
5. Share with implementers

2️⃣ CONTEXT ENGINEERING

What It Is

Providing the AI with **relevant information from THIS specific conversation** so it remembers what was said.

Analogy

Like taking notes during a meeting so you remember what was discussed.

In the Workshop

Part 3 (20 min): Implementers build conversation memory

Example Context:

```
Conversation so far:
User: "I need car insurance"
AI: "I'd be happy to help! What's your age?"
User: "I'm 28"
AI: "Great! What vehicle do you drive?"
User: "2020 Honda Civic"
AI: "Perfect! How many years have you been licensed?"
```

Code Location

`backend/main.py` (lines 132-142)

```python
# Get or create session
if session_id not in sessions:
    sessions[session_id] = {
        "messages": [],           # ← Context: conversation history
        "user_info": {},          # ← Context: extracted data
        "insurance_type": None,   # ← Context: what they need
```

```
            "quote_result": None,
            "knowledge_context": "",
            "next_action": "gather_info"
        }

    session = sessions[session_id]

    # Add user message to history
    session["messages"].append(HumanMessage(content=request.message))
```

When It's Used

- ✅ **Every message** in the conversation
- ✅ **Unique per user**
- ✅ **Defines WHAT was said**

Implementer Activity

1. Store conversation history
2. Track extracted information
3. Pass to LLM with each message
4. Agent remembers context

Why It Matters

Without context:

```
User: "I'm 28"
AI: "What's your age?"  ← Forgot user just said it!
```

With context:

```
User: "I'm 28"
AI: "Great! What vehicle do you drive?"  ← Remembers age
```

3️⃣ RAG (Retrieval Augmented Generation)

What It Is

Searching a knowledge base for facts and information to answer user questions accurately.

Analogy

Like looking up information in a company handbook before answering.

In the Workshop

Part 4 (20 min): Orchestrators write FAQs, Implementers build search

Example Knowledge Base:

```
Q: What is collision coverage?
A: Collision coverage pays for damage to YOUR vehicle when you hit
   another vehicle or object, regardless of who's at fault.

Q: What is comprehensive coverage?
A: Comprehensive coverage pays for damage to your vehicle from
   non-collision events like theft, vandalism, weather, or fire.

Q: How can I lower my premium?
A: Common ways include: bundling policies (10-25% off), maintaining
   a clean driving record (15-30% off), and increasing deductible.
```

Code Location

`backend/rag_system.py`

```python
# Knowledge base (from orchestrators!)
INSURANCE_KNOWLEDGE = {
    "auto_coverage": [
        "Collision coverage pays for damage to YOUR vehicle...",
        "Comprehensive coverage pays for damage from theft...",
        "Liability coverage pays for damage YOU cause..."
    ],
    "discounts": [
        "Bundling policies saves 10-25%...",
        "Clean driving record saves 15-30%..."
    ]
}

def search_knowledge(query: str, k: int = 2):
    """Search knowledge base for relevant information"""
    vectorstore = Chroma(...)
    results = vectorstore.similarity_search(query, k=k)
    return results
```

When It's Used

- ✅ **When user asks questions** ("What is X?")
- ✅ **Searches external knowledge**
- ✅ **Provides FACTS to the agent**

Orchestrator Activity

1. Write 10 insurance FAQs

2. Cover: coverage types, discounts, general
3. Keep answers clear
4. Share with implementers

Implementer Activity

1. Add FAQs to `INSURANCE_KNOWLEDGE`
2. Initialize vector database
3. Implement search function
4. Integrate with chat endpoint

How It Works

```
User asks: "What is collision coverage?"
         ↓
System detects question keyword ("what is")
         ↓
RAG searches knowledge base
         ↓
Finds: "Collision coverage pays for damage..."
         ↓
Adds to context for LLM
         ↓
LLM uses fact to answer accurately
```

🔄 How They Work Together

Example Conversation

User: "Hi, I need insurance"

What happens:

1. **Prompt Engineering**: Agent uses personality ("friendly, helpful")
2. **Context Engineering**: Stores message in history
3. **RAG**: Not triggered (no question asked)

Response: "I'd be happy to help! Are you looking for auto or home insurance?"

User: "Auto insurance. What's the difference between collision and comprehensive?"

What happens:

1. **Prompt Engineering**: Agent uses guideline ("explain clearly")
2. **Context Engineering**: Remembers user wants auto insurance
3. **RAG**: Searches knowledge base for "collision" and "comprehensive"

RAG finds:

```
- Collision: pays for damage to YOUR vehicle when you hit something
- Comprehensive: pays for damage from theft, weather, etc.
```

Response: "Great question! Collision coverage pays for damage to YOUR vehicle when you hit another vehicle or object. Comprehensive coverage pays for damage from non-collision events like theft, vandalism, or weather. Both are optional but recommended!"

User: "I'm 28, drive a 2020 Honda Civic"

What happens:

1. **Prompt Engineering**: Agent uses guideline ("ask 1-2 questions at a time")
2. **Context Engineering**: Stores age=28, vehicle=2020 Honda Civic
3. **RAG**: Not triggered (no question)

Response: "Perfect! How many years have you been licensed, and have you had any accidents or violations in the last 3 years?"

📊 Workshop Timeline

Part 2: Prompt Engineering (15 min)

Orchestrators: Design agent personality in AI Studio **Implementers**: Code `system_prompt.py` **Together**: Test personality

Part 3: Context Engineering (20 min)

Orchestrators: Create test cases **Implementers**: Build LangGraph with state management **Together**: Verify agent remembers context

Part 4: RAG (20 min)

Orchestrators: Write 10 FAQs **Implementers**: Implement vector search **Together**: Test knowledge retrieval

---

🎯 Key Differences

Prompt Engineering

- ✅ **Static**: Same for all users
- ✅ **Design time**: Created once
- ✅ **Defines**: WHO the agent is
- ✅ **Example**: "You are friendly and professional"

Context Engineering

- ✅ **Dynamic**: Unique per conversation
- ✅ **Runtime**: Updates every message

- ✅ **Defines**: WHAT was said
- ✅ **Example**: "User said age is 28"

RAG

- ✅ **On-demand**: Only when needed
- ✅ **Query time**: Searches when user asks
- ✅ **Defines**: FACTS from knowledge base
- ✅ **Example**: "Collision coverage definition"

💡 When to Use Each

Use Prompt Engineering When:

- Defining agent personality
- Setting conversation guidelines
- Specifying required information
- Establishing tone and style

Use Context Engineering When:

- Remembering user information
- Tracking conversation flow
- Avoiding repeated questions
- Maintaining state across messages

Use RAG When:

- User asks factual questions
- Need accurate, up-to-date information
- Have large knowledge base
- Want to avoid hallucinations

🧪 Hands-On Exercise

Test All Three

Scenario: User asks about insurance

Your Turn:

1. **Prompt Engineering**: Design personality

```
You are [NAME], a [ROLE] with [PERSONALITY].
```

2. **Context Engineering**: Track this conversation

```
User: "I need car insurance"
AI: "What's your age?"
User: "28"
AI: [Should remember age, ask next question]
```

3. **RAG**: Write a FAQ

```
Q: What is liability coverage?
A: [Your answer]
```

📚 Code Examples

1. Prompt Engineering

```
# backend/system_prompt.py
INSURANCE_AGENT_PROMPT = """
You are "Alex", an expert insurance agent.
Your personality: Friendly, professional, helpful
"""
```

2. Context Engineering

```
# backend/main.py
session = {
    "messages": [
        HumanMessage("I need insurance"),
        AIMessage("What's your age?"),
        HumanMessage("I'm 28")  # ← Context stored
    ],
    "user_info": {"age": 28}  # ← Extracted context
}
```

3. RAG

```
# backend/rag_system.py
INSURANCE_KNOWLEDGE = {
    "auto_coverage": [
        "Collision coverage pays for damage to YOUR vehicle..."
    ]
}

# Search when user asks
```

```
results = search_knowledge("What is collision coverage?")
# Returns: "Collision coverage pays for damage..."
```

✅ Summary Table

Feature	Prompt Eng	Context Eng	RAG
Frequency	Once	Every message	On-demand
Scope	All users	Per user	Per question
Storage	Code	Memory	Database
Created by	Orchestrators	System	Orchestrators + System
Purpose	Define behavior	Remember conversation	Provide facts
Example	"Be friendly"	"User is 28"	"Collision = ..."

🎓 Key Takeaways

1. **Prompt Engineering** = WHO the agent is (personality)
2. **Context Engineering** = WHAT was said (memory)
3. **RAG** = FACTS from knowledge base (search)
4. **All three work together** to create intelligent conversations
5. **Orchestrators design** prompts and knowledge
6. **Implementers build** context management and RAG
7. **Together** they create a smart agent

📖 Workshop Resources

- **Prompt Engineering**: `docs/PROMPT_ENGINEERING_GUIDE.md`
- **Context Engineering**: See LangGraph agent state
- **RAG**: `backend/rag_system.py`

Remember: These aren't competing approaches - they complement each other! 🚀