PROMPT ENGINEERING

Presented By : Dhruvi Mehta

INTRODUCTION OF PROMPT ENGINEERING

- Prompt engineering is the process of writing instructions to get the best output from an AI model. It's a key part of generative AI, which is changing how people interact with technology.

HISTORY OF PROMPT ENGINEERING

- **1950s–1980s**

  Early AI and NLP foundations (Turing Test, ELIZA, expert systems).

- **1990s–2010s**

  Machine learning and deep learning improved NLP (Word2Vec, rule-based systems).

- **2017s–2020s**

  Rise of large language models (Transformer architecture, GPT-1 to GPT-3).

- **2021s**

  Present: Prompt engineering becomes a key discipline (zero-shot, few-shot, CoT prompting, ChatGPT, GPT-4).

USES OF PROMPT ENGINEERING

1. Chatbot Development
   Utilizing prompt engineering to create conversational agents that understand human language effectively.

2. Content Generation
   Applying prompt engineering techniques to automatically generate articles, stories, or summaries.

3. Data Analysis
   Empowering data scientists to leverage prompts for extracting meaningful insights from large datasets.

4. Personalized Recommendations
   Using prompt engineering to enhance recommendation systems for tailored user experiences.

TYPES OF PROMPT

1. Instruction-Based Prompt – Direct command or request.
2. Open-Ended Prompt – Encourages exploration or discussion.
3. Contextual Prompt – Provides background information for a response.
4. Example-Based Prompt – Uses patterns or references to guide output.
5. Multi-Turn Prompt – Builds on previous interactions.
6. Chain-of-Thought Prompt – Encourages step-by-step reasoning.
7. Role-Based Prompt – Assigns a specific persona or expertise.

PLATFORMS SUPPORTING PROMPT EGG.

	Features	Use Cases	Access Method
OpenAI	API for LLMs	Chatbots	Web Interface
Google	BERT, T5	Search Optimization	Cloud-Based
Microsoft	Azure Integration	Code Assistance	API Access
Hugging Face	Transformers Library	Research, Development	Open Source

KEY FEATURES OF EFFECTIVE PROMPT ENGINEERING

- Clarity :- Ensure each prompt clearly specifies the desired outcome or response.
- Contextual Relevance :- Prompts should include relevant context to guide the AI's generation.
- Conciseness :- Keep prompts short while still conveying essential information.
- Direct Instruction :- Use imperative language to direct the AI towards specific tasks.
- Balanced Specificity :- Provide enough detail without overwhelming the AI to maintain flexibility.
- Iterative Testing :- Regularly refine prompts based on AI response quality and relevance.
- User Input :- Leverage user feedback to improve and adapt prompt strategies.

ADVANTAGES OF USING PROMPT ENGINEERING

1. Improves AI Output Quality

2. Enhances Customization and Personalization

3. Enhance Clarity

4. Boost Creativity

5. Enhance AI Debugging And code Assistance

6. Support Multi-Model AI(text, image, audio, video, code)

DISADVANTAGES OF USING PROMPT ENGINEERING

1. Dependency
2. Bias
3. Complexity
4. Context Limitation
5. Efficiency
6. OverFitting
7. User Expertise

BEST PRACTICES FOR WRITTING PROMPT

- Clarity
- Specificity
- context
- Conciseness
- Instruction
- Examples
- Testing

APPLICATIONS

AI-Powered Content Generation

- Writing articles, blogs, and essays
- Generating creative stories, poems, and scripts

Chatbots & Virtual Assistants

- Enhancing customer support responses
- Automating FAQs and troubleshooting

Healthcare & Medical Applications

- Assisting in medical report summarization
- Generating patient-friendly explanations of conditions

Data Analysis & Research

- Extracting insights from large datasets
- Generating reports and summaries

SOLUTION

Clear and Specific Prompts

Instead of: "Explain Nodejs"
Use: "Explain Nodejs Json Web Token example."
Solution: Be precise to get relevant responses.

Step-by-Step Prompts

Instead of: "Solve this math problem: 5x + 10 = 30."
Use: "Solve the equation 5x + 10 = 30 step by step."
Solution: Guide AI to break down complex tasks.

Format-Specific Prompts

Instead of: "Explain JavaScript."
Use: "Explain JavaScript closures with an example."
Solution: Specify output format (list, table, step-by-step).

Real-Time Debugging & Improvement

Example Prompt for Debugging:
"Find and fix errors in this JavaScript function. Explain the mistake."

FUTURE TRENDS

- Automated Prompt Optimization
- Multimodal Prompt Engineering
- AI-Powered Code Generation & Debugging
- Natural Language-Based Prompting
- Dynamic Prompt Chaining & Self-Correction

BACKEND CODE

Prompt_engi_demo > backend > JS app.js > ...

```javascript
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");
const { GoogleGenerativeAI } = require("@google/generative-ai");
const db = require("./database");
const mongoose = require("mongoose");

dotenv.config();
const app = express();
const PORT = process.env.PORT || 5000;

app.use(express.json());
app.use(cors());

const GEMINI_API_KEY = process.env.GEMINI_API_KEY;
const genAI = new GoogleGenerativeAI(GEMINI_API_KEY);


const promptSchema = new mongoose.Schema({
    prompt: String,
    strategy: String,
    response: String,
    createdAt: { type: Date, default: Date.now }
});
const Prompt = mongoose.model("Prompt", promptSchema);


app.post("/api/prompt", async (req, res) => {
    try {
        const { prompt, strategy } = req.body;
        if (!prompt || !strategy) {
            return res.status(400).json({ error: "Prompt and strategy are required" });
        }


        const formattedPrompt = `
Please respond in Markdown format (\`#\` for headings, \`*\` for bullet points).
```

Prompt_engi_demo > backend > JS app.js > ...

```javascript
app.post("/api/prompt", async (req, res) => {
        if (!prompt || !strategy) {
            return res.status(400).json({ error: "Prompt and strategy are required" });
        }


        const formattedPrompt = `
Please respond in Markdown format (\`#\` for headings, \`*\` for bullet points).
Request: "${strategy}: ${prompt}"
`;

        const model = genAI.getGenerativeModel({ model: "gemini-1.5-pro-latest" });
        const result = await model.generateContent([formattedPrompt]);

        const responseText = result.response.candidates[0].content.parts[0].text;


        const newPrompt = new Prompt({ prompt, strategy, response: responseText });
        await newPrompt.save();

        res.json({ result: responseText });
    } catch (error) {
        console.error("Error generating AI response:", error);
        res.status(500).json({ error: "Failed to process the request", details: error.message });
    }
});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

app.js

DATABASE AND .ENV

```
const mongoose = require("mongoose");
const dotenv = require("dotenv");

dotenv.config();

mongoose.connect(process.env.MONGO_URL)
.then(() => console.log("Connected to MongoDB"))
.catch((err) => console.error("MongoDB Connection Error:", err));

module.exports = mongoose;
```

```
MONGO_URL=mongodb://localhost:27017/prompt
GEMINI_API_KEY=AIzaSyA2O6Opgs8gSXvkUTM_DnkfBaJSKVI23uk
```

FRONTEND

Prompt_engi_demo > frontend > src > components > JS Chatbot.jsx > Chatbot > handleSend

```jsx
import React, { useState } from "react";
import axios from "axios";
import ReactMarkdown from "react-markdown"; // For rendering Markdown

const Chatbot = () => {
  const [prompt, setPrompt] = useState("");
  const [strategy, setStrategy] = useState("Detailed Explanation");
  const [messages, setMessages] = useState([]);
  const [loading, setLoading] = useState(false);

  const handleSend = async () => {
    if (!prompt.trim()) return;

    const userMessage = { sender: "User", text: prompt };
    setMessages([...messages, userMessage]);
    setPrompt("");
    setLoading(true);

    try {
      const response = await axios.post("http://localhost:5000/api/prompt", {
        prompt,
        strategy,
      });

      const botMessage = { sender: "Bot", text: response.data.result };
      setMessages((prev) => [...prev, botMessage]);
    } catch (error) {
      console.error("Error:", error);
      setMessages((prev) => [...prev, { sender: "Bot", text: "Error processing request." }]);
    }

    setLoading(false);
  };

  return (
    <div className="chat-container">
      <div className="chat-box">
```

```jsx
  return (
    <div className="chat-container">
      <div className="chat-box">
        {messages.map((msg, index) => (
          <div key={index} className={`message ${msg.sender === "User" ? "user" : "bot"}`}>
            <b>{msg.sender}:</b>
            <ReactMarkdown>{msg.text}</ReactMarkdown> {/* Renders Markdown */}
          </div>
        ))}
        {loading && <div className="bot loading">Bot is typing...</div>}
      </div>

      <div className="input-container">
        <input
          type="text"
          value={prompt}
          onChange={(e) => setPrompt(e.target.value)}
          placeholder="Enter your prompt..."
          className="chat-input"
        />
        <select className="chat-select" value={strategy} onChange={(e) => setStrategy(e.target.value)}>
          <option value="Detailed Explanation">Detailed Explanation</option>
          <option value="Short Summary">Short Summary</option>
          <option value="Key Points">Key Points</option>
        </select>
        <button className="chat-send" onClick={handleSend} disabled={loading}>Send</button>
      </div>
    </div>
  );
};

export default Chatbot;
```

App.jsx