INFO 7375 – Prompt Engineering and Generative AI Fall Final Exam 2024

Introduction to Large Language Models (LLMs)

1.1 What is an LLM?

1.2 Pre-training and Fine-Tuning Overview

1.3 Challenges in LLM Development

Comparison: C++ vs. Prompt Engineering

2.1 Structural Differences Between Programming and Prompt Design

2.2 Procedural Logic vs. Contextual Logic in LLMs

2.3 Syntax and Optimization Variations

Prompt Fundamentals

3.1 What is Intent, Flow, and Dialog?

3.2 Context, Specificity, and Scaffolding

Prompt Patterns

4.1 Overview of Prompt Patterns

4.2 Advanced Prompt Patterns for Optimization

Advanced Techniques in Prompt Engineering

5.1 Socratic Method-Based Teaching for Effective Prompt Design

5.2 Optimized Use of Generative AI

5.3 How to Speak Bot: Prompt Engineering for Customization

Dynamic Generative Content

6.1 Techniques for Generating Dynamic and Context-Aware Content

6.2 Applications in Conversational Agents and Content Personalization

Advanced Techniques in Fine-Tuning LLMs

7.1 Transfer Learning Techniques

7.2 Fine-Tuning for Domain-Specific Tasks

7.3 Evaluating Fine-Tuned Models

7.4 Challenges in Fine-Tuning

LLM Hallucination

8.1 What is LLM Hallucination?

8.2 Identifying and Mitigating Hallucination Risks

Retrieval-Augmented Generation (RAG)

9.1 What is RAG?

9.2 Integration with Vector Databases

9.3 Applications of RAG in Real-World Scenarios

Vector Databases and Text Embeddings

10.1 Basics of Vector Databases

10.2 Role of Text Embeddings in Retrieval Tasks

10.3 Applications of Vector Databases in LLMs

Transformer Architecture and Neural Networks

11.1 Components of Transformer Models

- Self-Attention Mechanism
- Multi-Head Attention
- Feedforward Neural Networks

11.2 Positional Encoding in Transformers

- Necessity of Positional Encoding
- Encoding Techniques

11.3 Applications of Transformers in LLMs

- Role in Generative AI
- Advancements Over Previous Architectures

Combining Old School and New

12.1 Hybrid Approaches: Combining Traditional AI Techniques with LLMs

Cheat Sheet

INFO 7375 – Large Language Models & Prompt Engineering Cheat Sheet

---

1. Large Language Models (LLMs) Fundamentals

1.1 What is an LLM?

- **Definition**: Neural networks trained on vast text data to understand and generate human-like text.
- **Key Capabilities**: Text generation, translation, summarization, code writing, and creative writing.
- **Example**: GPT-4 can process ~50 pages of text in a single prompt and generate coherent responses.

1.2 Pre-training and Fine-Tuning

- **Pre-training**: Initial training on a broad dataset to learn language patterns.
  - Example: Training on internet text data to understand general language constructs.
- **Fine-Tuning**: Specialized training for domain-specific tasks.
  - Example: Fine-tuning GPT-3 on medical texts for healthcare applications.

1.3 Challenges

- **Computational Resources**: Requires significant GPU/TPU power.
- **Data Quality**: Biased data can lead to biased outputs.
- **Hallucination**: Risk of generating false or inconsistent information.

---

2. C++ vs. Prompt Engineering Comparison

2.1 Structural Differences

- **C++ Example**:

cpp

Copy code

```
if (condition) {
    executeFunction();
} else {
    alternativeFunction();
}
```

- **Prompt Engineering Equivalent**:

plaintext

Copy code

```
If the input matches [condition], respond with [specific
output].
Otherwise, provide [alternative response]. Consider the context
that [relevant details].
```

2.2 Procedural vs. Contextual Logic

- **C++**: Explicit, step-by-step instructions.
- **Prompt Engineering**: Natural language, context-driven instructions.

2.3 Optimization

- **C++ Optimization**:

cpp

Copy code

```
vector<int> nums(1000000);
nums.reserve(1000000); // Pre-allocate memory
```

- **Prompt Optimization**:

    - Initial: "Write a story."
    - Optimized: "Write a 500-word story about a detective in cyberpunk Tokyo, focusing on noir elements and including detailed sensory descriptions."

---

3. Prompt Fundamentals

3.1 Intent, Flow, and Dialog

- **Intent Example**:
    1. **Bad**: "Tell me about cars."
    2. **Good**: "Explain the evolution of electric vehicles from 2010 to 2024, focusing on Tesla's impact."
- **Flow Example**:
    1. Set Context: "You are an expert in renewable energy."
    2. Define Task: "Explain solar panel efficiency."
    3. Specify Output: "Use technical terms with layman explanations."

3.2 Context and Specificity

- **Scaffolding Example**:
    1. Background: "Assuming knowledge of basic Python."
    2. Task: "Create a function to calculate Fibonacci numbers."
    3. Constraints: "Optimize for space complexity."
    4. Output: "Include comments and example usage."

---

4. Advanced Prompt Patterns

4.1 Common Patterns

- **Chain-of-Thought**:

plaintext

Copy code

```
Solve this step by step:

1. First, consider...
```

```
2. Then, analyze...
3. Finally, conclude...
```

- **Few-Shot Learning**:

plaintext

Copy code

```
Example 1: Input -> Output
Example 2: Input -> Output
Now solve: New Input
```

4.2 Optimization Patterns

- **Temperature Control**:
    - For creative writing: "Use more creative and varied language."
    - For technical writing: "Provide precise, factual information."

---

5. Dynamic Content Generation

5.1 Contextual Generation

- **Example**:
    - Base Prompt: "Write a product description."
    - Contextual Additions:
        - Audience: "For tech-savvy millennials."
        - Tone: "Using casual, friendly language."
        - Features: "Emphasizing sustainability."

---

6. RAG Implementation

6.1 Basic RAG Structure

- **Pseudo-Code**:

python

Copy code

```
def RAG_system(query):
    relevant_docs = vector_db.search(query)
    enhanced_prompt = combine(query, relevant_docs)
    return llm.generate(enhanced_prompt)
```

---

7. Vector Databases

7.1 Text Embeddings

- **Example**:

python

Copy code

```
# Creating embeddings
text = "Example sentence"
embedding = model.encode(text)
vector_db.store(embedding)
```

---

8. Transformer Architecture

8.1 Self-Attention Mechanism

- **Simplified Example**:

python

Copy code

```
def self_attention(query, key, value):
    attention_weights = softmax(query @ key.transpose())
    return attention_weights @ value
```

Best Practices Checklist

- Always specify the desired format and structure.
- Include examples for complex tasks.
- Break down complex prompts into smaller components.
- Use clear and unambiguous language.
- Test prompts with various inputs.
- Include error-handling instructions.
- Specify constraints and limitations.

Note

This cheat sheet consolidates key concepts and practical examples to provide a comprehensive understanding of LLMs and prompt engineering. Practical experimentation and iterative refinement are essential for mastery.