CodeLegend1011

Prompt Engineering

- **Basic LLM Concepts:**
  - **What are LLMs?**

    LLMs, or Large Language Models, are advanced AI systems trained on vast amounts of text data to understand and generate human-like text. They have diverse applications, including language translation and content generation. Examples include GPT models developed by OpenAI. They use transformer architectures and undergo pre-training and fine-tuning. While they excel in natural language understanding tasks, there are ethical concerns regarding their potential misuse. Despite their capabilities, they still face challenges in understanding context and maintaining ethical standards.

  - **Types of LLMs:**

    GPT (Generative Pre-trained Transformer): Developed by OpenAI, GPT models are among the most well-known LLMs. Examples include GPT-2 and GPT-3. These models are pre-trained on vast amounts of text data and fine-tuned for specific tasks.

    BERT (Bidirectional Encoder Representations from Transformers): Introduced by Google, BERT models are designed to understand the context of words in a sentence. They are pre-trained on large corpora and have been widely used in natural language understanding tasks.

    XLNet: XLNet is another variant of the transformer architecture that leverages permutation language modeling. It considers all possible permutations of words in a sentence during pre-training, leading to improved performance on various tasks.

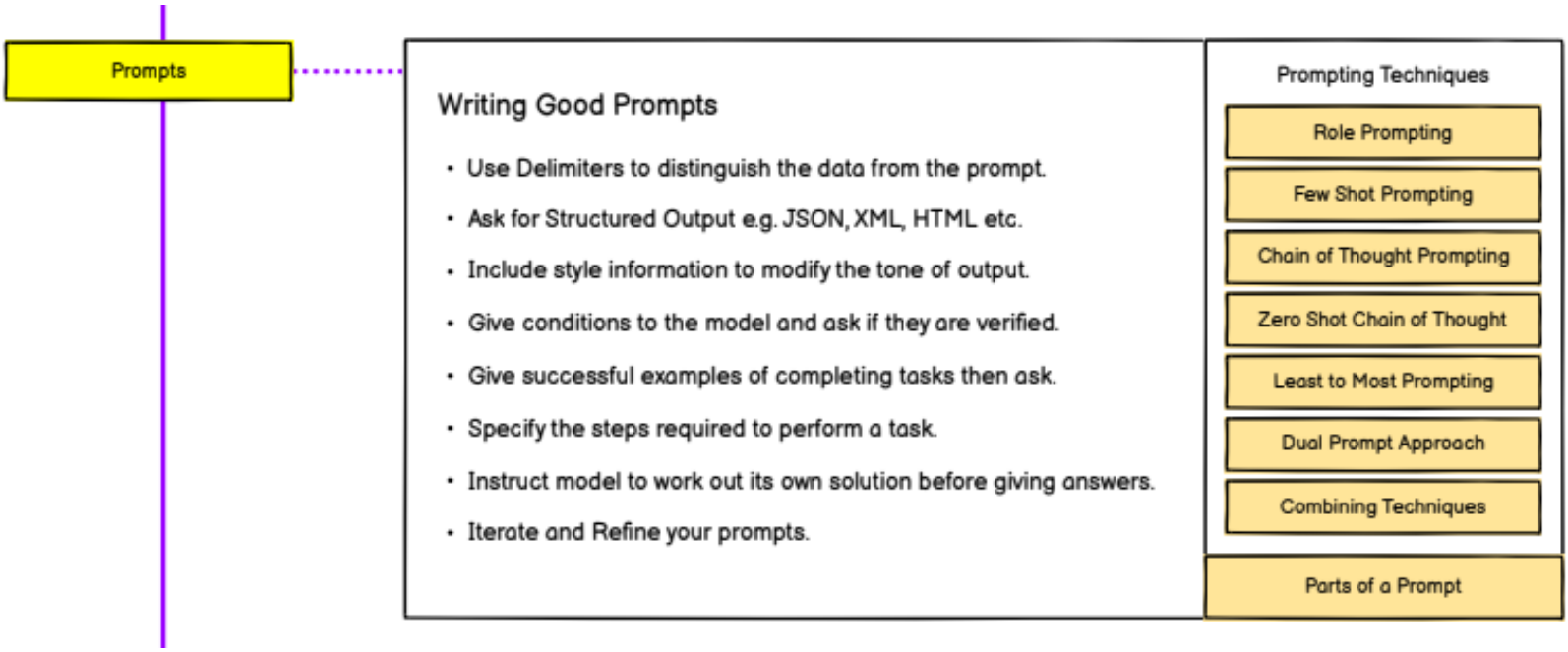
    T5 (Text-To-Text Transfer Transformer): Developed by Google, T5 treats all NLP tasks as text-to-text tasks, where both the input and output are in natural language. It achieves state-of-the-art results on a wide range of NLP benchmarks.

CodeLegend1011

- **How are LLM Built?**

1. **Data Collection:** LLMs require vast amounts of text data to train effectively. This data is collected from various sources, including books, articles, websites, and other written material.
2. **Pre-processing:** The collected text data undergoes pre-processing, which includes tasks such as tokenization, where the text is split into individual words or subwords, and normalization, where the text is standardized to a common format.
3. **Model Architecture:** LLMs are built using transformer architectures, which consist of multiple layers of self-attention mechanisms. These architectures facilitate parallel processing and allow the model to capture long-range dependencies in the input text.
4. **Training:** LLMs are trained using unsupervised learning techniques, where the model learns to predict the next word in a sequence of text given the preceding words. This pre-training phase typically involves large-scale distributed computing resources and can take several days or weeks to complete.
5. **Fine-tuning:** After pre-training, LLMs are fine-tuned on specific tasks or domains to improve their performance. Fine-tuning involves training the model on task-specific datasets with annotated examples, adjusting the model's parameters to better suit the target task.
6. **Deployment:** Once trained and fine-tuned, LLMs can be deployed for various applications, including text generation, language translation, sentiment analysis, and more. They are typically deployed as APIs or integrated into software applications to provide natural language processing capabilities.

CodeLegend1011



=>Real World Usage Examples:

1. **Structured Data:** LLMs can extract and interpret structured data from unstructured text, aiding in tasks like data analysis and information retrieval.
2. **Inferring:** They excel at inferring missing information from context, useful in completing forms, understanding ambiguous statements, etc.
3. **Writing Emails:** LLMs can assist in composing emails by suggesting content based on context or generating responses to common queries.
4. **Coding Assistance:** Developers use LLMs for code completion, debugging assistance, and generating documentation.
5. **Study Buddy:** LLMs serve as virtual study partners, helping students understand complex topics, generating summaries, and answering questions.
6. **Designing Chatbots:** LLMs power chatbots across customer service, healthcare, and education sectors, providing human-like interaction.

CodeLegend1011

=>Pitfalls of LLMs:

1. **Citing Sources:** LLMs may not always cite sources, leading to potential plagiarism issues in generated content.
2. **Bias:** They can perpetuate biases present in training data, affecting decisions and perpetuating stereotypes.
3. **Hallucinations:** LLMs sometimes generate nonsensical or misleading content due to imperfect training or prompt manipulation.
4. **Math:** While proficient in basic math, LLMs may struggle with complex mathematical reasoning and precision.
5. **Prompt Hacking:** Adversaries can manipulate prompts to produce unintended or harmful outputs.

=>Improving Reliability:

1. **Prompt Debiasing:** Implement strategies to mitigate biases in prompts and training data.
2. **Prompt Ensembling:** Combine multiple prompts to reduce the impact of individual biases or errors.
3. **LLM Self Evaluation:** Develop mechanisms for LLMs to assess the quality and reliability of their own outputs.
4. **Calibrating LLMs:** Adjust model parameters to improve accuracy and reduce errors in generated text.

CodeLegend1011

=>LLM Settings:

1. **Temperature:** Controls the randomness of generated text, affecting creativity vs. coherence.
2. **Top P:** Determines the probability threshold for word selection, influencing diversity and relevance.
3. **Other Hyperparameters:** Include parameters related to model architecture, training duration, and fine-tuning strategies.

=>Prompt Hacking:

1. **Prompt Injection:** Injecting additional context to steer LLM outputs in desired directions.
2. **Prompt Leaking:** Accidentally revealing sensitive information through prompt construction.
3. **Jailbreaking:** Unauthorized access to LLM internals, potentially for malicious purposes.
4. **Defensive Measures:** Employ robust prompt design and model training to counteract adversarial attacks.
5. **Offensive Measures:** Counteract adversarial attacks with robust prompt design and model training.

=>Image Prompting:

1. **Style Modifiers:** Adjust language style or tone to match specific communication requirements.
2. **Quality Boosters:** Integrate mechanisms for post-generation quality assessment and refinement.
3. **Weighted Terms:** Assign priority weights to certain words or phrases to guide LLM outputs.

CodeLegend1011

=>Elements of Prompt: A prompt for AI models typically contains four elements:

1. **Instructions**
2. **Context**
3. **Input/Data**
4. **Output/Indicator**
5. **Example**

=>Points to be Noted:-

- Precise Instructions: Provide specific commands like “Write”, “Classify”, “Summarise”, “Translate”, ”Order”, etc
- Be Specific: About details , required output, length , limitations
- Important Libraries: openai, transformers (Hugging Face AI)
- Other Major Elements of a Prompt: Text Classification, Conversation, Code Generation, Reasoning.
- Self-Consistency, Factual, General Knowledge
- Single Prompt Approach & Double Prompt Approach

=>Zero-Shot Prompting:

Zero-shot prompting refers to a technique used with large language models where the model is prompted to generate text or perform a task without explicit training on that specific prompt or task. Instead of being fine-tuned on a specific prompt or task, the model is expected to generalize its knowledge and capabilities to respond appropriately based on the given prompt.

=> Few-Shot Prompting:

Few-shot prompting is a technique used with large language models where the model is trained on a limited amount of data (a few examples or shots) for a specific prompt or task. Unlike traditional fine-tuning approaches that require extensive training data, few-shot prompting enables the model to adapt quickly to new tasks or prompts with minimal examples.

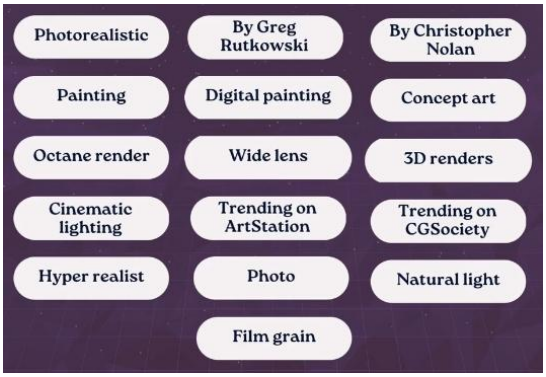
CodeLegend1011

=> Chain-of-Thought Prompting (CoT prompting):

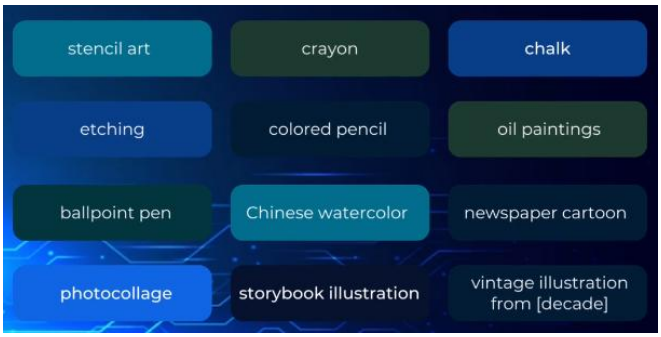
Chain of thought prompting involves guiding a language model through a series of steps in its reasoning process, akin to leading it down a logical pathway. This technique is used to steer the model towards generating coherent and contextually relevant responses by structuring the prompts in a sequential manner. It helps in shaping the model's thought process and ensuring that the generated outputs follow a logical flow. This method is particularly useful in scenarios where the desired output requires a step-by-step reasoning or argumentation, such as in generating explanations, solving multi-step problems, or composing structured essays.

=>Image Prompting:

- Style-Modifiers:

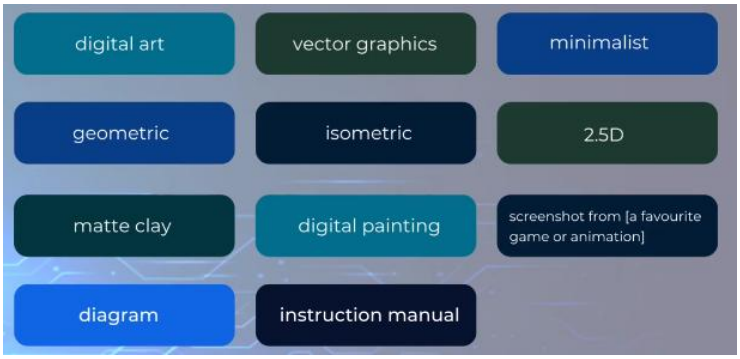


- Quality Boosters: Amazing, Beautiful, Majestic, etc
- Attention to detail: Descriptive objective; Colourful, Swirling, Happy, Dramatic, Geometric, Abstract, etc
- Type of Filming, Camera angle, Lighting
- Art Medium Styles:



CodeLegend1011

- Digital Styles:



- Retro Styles:

- Photographic Styles:

CodeLegend1011

- Object Making:

- Content Type > Description> Style> Composition
- Dall-E, DeepAI, Pixlr.com, Adobe Firefly, Midjourney, Stable Diffusion, etc
- Weight terms: Example Prompt- A planet in space:10 | bursting with color red, blue, and purple:4 | aliens: -10 | 4K, high quality.
- --ar: aspect ratio, --c: chaos value, --seed: arbitrary seed value, --q:quality, --s: stylize value, --version, --sameseed: noise for all images, --title, --video
- DOCS: Direct-Obvious-Clear-Specific
- Define the prompt and prompt engineering concept; State the elements of a prompt; Explain how to design a prompt; Indicate how to set up your environment and install the necessary libraries; Discuss basic prompting methods; Explain advanced prompting methods, such as Zero-Shot Prompting and Few-Shot Prompting; Discuss Dall-E's abilities and how to write prompts for Dall-E; Describe the Outpainting feature of Dall-E; State how to generate a desired image in Stable Diffusion; Identify effective prompting using ChatGPT; List the general guidelines for OpenAI Prompt Engineering; Outline Prompt Engineering best practices